11 Speichern von Daten

Einer der wichtigsten Einsatzbereiche von Pocket PCs ist die mobile Datenerfassung. Doch was nützt es, wenn der Pocket PC die Daten nicht speichern kann? Daher wollen wir uns in den nächsten Kapiteln mit den verschiedenen Möglichkeiten der Datenspeicherung auseinander setzen, die ein Pocket PC bietet: angefangen bei einfachen Textdateien, über die Verwendung der Registrierungsdatenbank bis hin zu komplexen SQL-Datenbanken.

Jede dieser Methoden hat bestimmte Vor- und Nachteile und eignet sich damit jeweils für einen anderen Einsatzzweck. Wollen Sie unstrukturierte Daten wie z.B. längere Texte speichern, eignen sich sequentielle Textdateien dazu am besten. Wenn Sie aber nur ein paar Konfigurationsdaten für eine Anwendung speichern wollen, bietet sich die Registrierdatenbank an. Geht es um das Sichern von strukturierten Daten – wie z.B. Adressdaten –, sind Datenbanken die erste Wahl.

Doch dazu später mehr. Befassen wir uns vorab zunächst mit der Verwendung von Dateien.

11.1 Dateien lesen und schreiben

Zum Arbeiten mit Dateien benötigen Sie das **Microsoft CE File System Control** 3.0, das Sie wie gewohnt über das Dialogfeld COMPONENTS in Ihr Projekt einbinden können.

Hinweis zur Buch-CD:

Die Beispielprojekte zu diesem Thema finden Sie auf der Buch-CD im folgenden Verzeichnis:

`\Beispiele\Kapitel 11\Dateien lesen und schreiben\`

Das MICROSOFT CE FILE SYSTEM CONTROL besteht eigentlich aus zwei Steuerelementen: Das erste ist das `FileSystem`-Steuerelement selbst, das verschiedene Funktionen zum Arbeiten mit Verzeichnissen und Dateien zur Verfügung stellt. Das andere ist das `File`-Steuerelement, das Methoden bereitstellt, um Dateien zu öffnen und schließen sowie Daten zu schreiben und zu lesen.

11.1.1 Textdateien

Wir wollen nun eine kleine Anwendung entwickeln, die Texte speichern und laden kann. Erzeugen Sie dazu ein neues Projekt, binden Sie das Microsoft CE File System Control 3.0 in dieses Projekt ein und platzieren Sie ein `File`-Steuerelement auf dem Formular der

Anwendung. Dieses Control erhält automatisch den Namen `File1`. Über dieses Objekt sind die Dateifunktionen erreichbar.

Weiterhin benötigen wir auf der Maske noch ein großes Textfeld (`txtNotiz`) sowie zwei Schaltflächen (`btnSpeichern` und `btnLaden`).

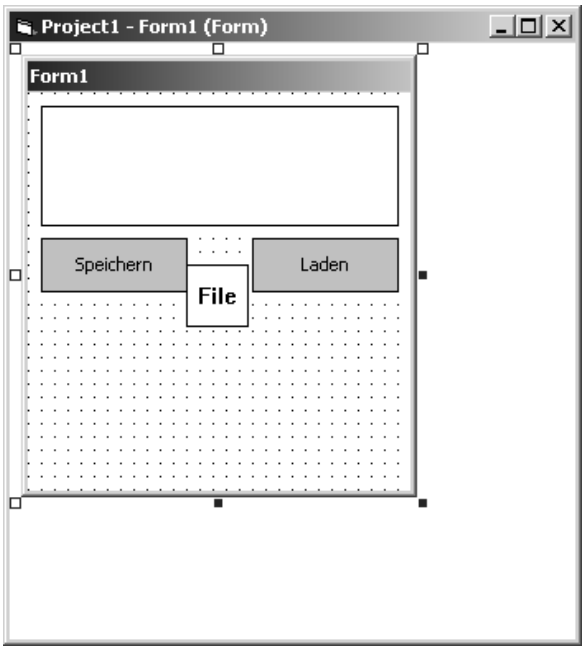

Abb. 11.1 Formular mit File-Steuerelement im Entwurfsmodus

Nun müssen Sie lediglich noch den folgenden Quelltext zu den beiden Schaltflächen hinterlegen:

```
Option Explicit

Private Sub btnLaden_Click()
    File1.Open "Testnotiz.txt", fsModeInput, fsAccessRead
    txtNotiz.Text = File1.LineInputString
    File1.Close
End Sub

Private Sub btnSpeichern_Click()
    File1.Open "Testnotiz.txt", fsModeOutput, fsAccessWrite
    File1.LinePrint txtNotiz.Text
    File1.Close
End Sub
```

```
Private Sub Form_OKClick()
    App.End
End Sub
```

Fertig ist die einfachste Variante einer sehr simplen „Textverarbeitung“.

Wenn der Anwender nun nach Eingabe eines Textes in das Textfeld die SPEICHERN-Schaltfläche anklickt, wird zuerst die Datei über die Methode `Open` geöffnet. Der Einfachheit halber wird erst einmal ein konstanter Dateiname („`Testnotiz.txt`“) verwendet. Die Parameter sind Konstanten und steuern die Art, in der die Datei geöffnet wird. `fsModeOutput` bewirkt, dass eine Textdatei zum Schreiben geöffnet wird und der folgende Parameter `fsAccessWrite`, dass in diese Datei auch wirklich nur geschrieben werden kann.

Mit der Methode `LinePrint` wird der Inhalt des Textfeldes `txtNotiz` in die geöffnete Datei geschrieben, die anschließend über die Anweisung `Close` geschlossen wird.

Der Anwender kann jetzt das Textfeld löschen oder die Applikation über den OK-Knopf beenden und neu starten. Nach einem Klick auf den LADEN-Button erscheint der vorher gesicherte Text wieder im Textfeld. Der interne Ablauf der Routine ist ähnlich wie beim Speichern:

Zuerst wird die Datei geöffnet. Die Parameter `fsModeInput` und `fsAccessRead` sorgen dafür, dass dieses Öffnen für das Lesen einer Textdatei gilt. Dann wird über die Methode `File1.LineInputString` der Text (oder genauer eine Textzeile) eingelesen und als Textinhalt an das Textfeld `txtNotiz` übergeben. Anschließend wird die Datei wieder geschlossen.

Sie werden sich nun vielleicht fragen, wie man bei mehrzeiligen Textboxen dafür Sorge tragen kann, dass auch alle Zeilen gelesen und gespeichert werden. Dazu wollen wir die Anwendung noch etwas erweitern.

Zuerst muss die Eigenschaft `MultiLine` der TextBox auf `true` gesetzt werden, damit auch mehrzeilige Eingaben angenommen werden. An der Routine zum Speichern brauchen Sie weiter nichts zu ändern. Hier wird ohnehin der ganze Inhalt der TextBox an die Methode `LinePrint` übergeben, sodass der gesamte Text – unabhängig von der Zeilenzahl – gesichert wird.

Die Methode `LineInputString` liest den Text jedoch nur bis zum nächsten Zeilenumbruch. Daher sind für die Routine zum Laden folgende Anpassungen nötig:

```
Private Sub btnLaden_Click()
    Dim Inhalt As String

    File1.Open "Testnotiz.txt", fsModeInput, fsAccessRead
    Inhalt = ""
    While Not File1.EOF
        Inhalt = Inhalt + File1.LineInputString + vbCrLf
```

```
    Wend
    txtNotiz.Text = Inhalt
    File1.Close
End Sub
```

Zuerst wird eine lokale Variable deklariert, in der wir die verschiedenen Textzeilen „sammeln“, bevor sie in die TextBox geschrieben werden. Anschließend wird die Datei wie gehabt zum Lesen geöffnet und die Variable `Inhalt` geleert.

Nun wird so oft jeweils eine Zeile gelesen und der Variablen hinzugefügt, bis das Ende der Datei erreicht ist. Das Dateiende wird über die Methode `EOF` (**E**nd **O**f **F**ile) des File Controls abgefragt. Nach jeder gelesenen Zeile wird außerdem ein Zeilenumbruch (`vbCrLf`) zur Variablen hinzugefügt, da dieses Zeichen ja von der Funktion `LineInputString` als Begrenzungszeichen erkannt und daher nicht im Funktionswert zurückgegeben wurde.

Nachdem das Dateiende erreicht ist, wird der Variableninhalt in die TextBox gesetzt und die Datei geschlossen.

In diesem Zusammenhang erscheint noch eine zweite Erweiterung des Programms sinnvoll: Momentan wird die Datei bei jedem Klicken auf die SPEICHERN-Schaltfläche überschrieben. Sie können dem Anwender aber auch die Möglichkeit geben, eine bestehende Datei zu erweitern. Fügen Sie dazu eine zweite SPEICHERN-Schaltfläche in das Formular ein und hinterlegen Sie diesem die folgende Ereignisprozedur:

```
Private Sub btnSpeichern2_Click()
    File1.Open "Testnotiz.txt", fsModeAppend, fsAccessWrite
    File1.LinePrint txtNotiz.Text
    File1.Close
End Sub
```

Auf den ersten Blick gleicht diese Prozedur der bereits bekannten. Es gibt allerdings einen kleinen Unterschied: Anstelle des Parameters `fsModeOutput` wurde hier `fsModeAppend` verwendet, wodurch die Datei nicht neu angelegt, sondern erweitert wird (append = englisch für „anhängen“).

Je nachdem, welche der beiden SPEICHERN-Schaltflächen der Benutzer verwendet, wird die Textdatei entweder neu erzeugt oder um den eingegebenen Text erweitert.

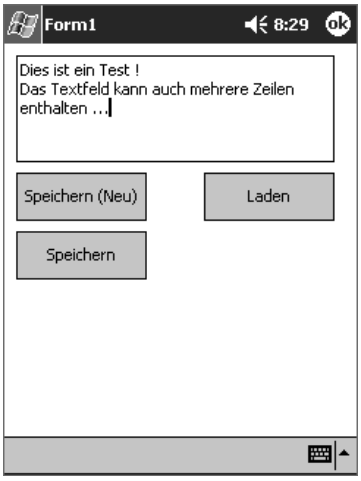

Abb. 11.2 TextBox mit mehrzeiligem Text

Unicode-Textdateien

Einen Sonderfall der Textdateien stellen die Unicode-Dateien dar. Im Prinzip handelt es sich dabei um ganz normale Textdateien, nur mit dem Unterschied, dass ein Zeichen hier nicht durch ein Byte, sondern durch zwei Bytes repräsentiert wird. Daraus ergibt sich ein größerer Vorrat an verfügbaren Zeichen, wodurch internationale Sonderzeichen besser abbildbar sind. Im Normalfall sollten Sie damit allerdings nicht oft in Berührung kommen, daher will ich nur kurz darauf eingehen, wie Sie Unicode-Dateien lesen können.

Um mit der bisherigen Anwendung auch Unicode-Dateien lesen zu können, müssen Sie lediglich die Ereignisprozedur `btnLaden_Click` ändern:

```
Private Sub btnLaden_Click()
    Dim Inhalt As String
    Dim byte1, byte2
    Dim bolUnicode

    File1.Open "Testnotiz.txt", fsModeInput, fsAccessRead
    byte1 = File1.Input(1)
    byte2 = File1.Input(1)

    Inhalt = ""
    If Asc(byte1) = 255 And Asc(byte2) = 254 Then
        bolUnicode = True
    Else
        bolUnicode = False
        Inhalt = Inhalt + byte1 + byte2
```

```
    End If
    While Not File1.EOF
        byte1 = File1.Input(1)
        If bolUnicode = True Then
            byte2 = File1.Input(1)
        End If
        Inhalt = Inhalt + byte1
    Wend
    txtNotiz.Text = Inhalt
    File1.Close
End Sub
```

Nach der Deklaration von vier Variablen wird die Datei wie gehabt geöffnet.

Nun werden aber erst einmal zwei Zeichen über die Methode `Input` aus der Datei gelesen. Die Methode `Input` liest die über den Parameter angegebene Anzahl von Zeichen (in diesem Fall jeweils eins) aus der Datei.

Anschließend wird verglichen, ob die ASCII-Werte der beiden Zeichen den Zahlen 255 und 254 entsprechen (diese Zeichenfolge wird am Dateianfang benutzt, um Dateien als Unicode-Dateien zu kennzeichnen). In jedem Fall wird die Variable `bolUnicode` so gesetzt, dass aus ihr ablesbar ist, ob es sich um eine Unicode-Datei handelt oder nicht.

Handelt es sich um keine Unicode-Datei, so werden die beiden gelesenen Zeichen zwischengespeichert, da sie in diesem Fall schon einen Teil des eigentlichen Textes enthalten.

Anschließend folgt wieder eine `WHILE`-Schleife, durch die der Text gelesen wird. Hier aber nicht zeilen-, sondern zeichenweise. Handelt es sich um eine Unicode-Datei, wird nach jedem Zeichen noch ein weiteres Zeichen gelesen, das ignoriert wird. Somit werden die Sonderzeichen, die eine Unicode-Datei zusätzlich darstellen kann, zwar nicht interpretiert, die normal darstellbaren Zeichen aber korrekt eingelesen.

Am Ende der Routine wird der zwischengespeicherte Text wieder in die TextBox gesetzt und die Datei geschlossen.

11.1.2 Binärdateien

Im Gegensatz zu Textdateien können Sie in Binärdateien Daten jeder Art ablegen. Dafür benötigen Sie aber wiederum einen anderen Dateiaufbau. So werden intern neben den eigentlichen Nutzdaten noch weitere Informationen in der Datei gespeichert. Dabei werden mit jeder Schreiboperation zuerst zwei Bytes geschrieben, die den Typ der folgenden Daten kennzeichnen. Bei Daten vom Typ `String` folgen zwei weitere Bytes, welche die Länge der Zeichenkette angeben. Anschließend folgen dann die eigentlichen Daten. Das Speichern dieser Zusatzinformationen (Datentyp und Stringlänge) erfolgt jedoch implizit, Sie brauchen sich also nicht selbst darum zu kümmern.

Sie können den Umgang mit Binärdateien ausprobieren, indem Sie das bisherige Beispiel verwenden und lediglich die Ereignisprozeduren für die beiden Schaltflächen anpassen:

```
Option Explicit

Private Sub btnLaden_Click()
    Dim Zeichen

    File1.Open "Testnotiz.bin", fsModeBinary, fsAccessRead

    While Not File1.EOF
        File1.Get Zeichen
        txtNotiz.Text = txtNotiz.Text + Zeichen
    Wend

    File1.Close
End Sub

Private Sub btnSpeichern_Click()
    Dim Zeichen
    Dim i

    File1.Open "Testnotiz.bin", fsModeBinary, fsAccessWrite

    For i = 1 To Len(txtNotiz.Text)
        File1.Put Mid(txtNotiz.Text, i, 1)
    Next

    File1.Close
End Sub

Private Sub Form_OKClick()
    App.End
End Sub
```

Hierbei gibt es im Vergleich zu den Textdateien zwei wesentliche Unterschiede:

Zum einen wird die Datei im Modus `fsModeBinary` geöffnet (sowohl zum Lesen als auch zum Schreiben von Daten). Der zweite Unterschied liegt darin, dass nun die Methoden `Put` und `Get` verwendet werden, um jeweils ein einzelnes Zeichen zu schreiben bzw. zu lesen.

Da es bei Binärdateien keine Rolle spielt, welchen Variablentyp Sie speichern, kann man auch anstelle des Speicherns Zeichen für Zeichen, das ganze Textfeld als einen String in der Datei ablegen. Dazu ersetzen Sie die `For`-Schleife in der Prozedur `btnSpeichern_Click` einfach durch die folgende Anweisung:

```
File1.Put txtNotiz.Text
```

Somit wird mit einer einzigen `Put`-Anweisung das gesamte Textfeld als String gespeichert. Die Routine `btnLaden_Click` müssen Sie dabei nicht anpassen, da das erste `Get` nun automatisch den gesamten Text liest.

Der Vorteil gegenüber der ersten Variante liegt darin, dass auf diese Art nur einmal Zusatzinformationen (Datentyp und Stringlänge) geschrieben werden, während bei der ursprünglichen Variante für jedes Byte an Nutzdaten drei Bytes in die Datei geschrieben werden (2 Byte für den Datentyp + 1 Byte Nutzdaten). Die Datei wird damit unnötig groß.

Eine weitere Besonderheit liegt darin, dass beim wiederholten Schreiben in die Datei die Daten nicht angehängt, sondern überschrieben werden. Das Überschreiben findet allerdings nicht wie bei der Textdatei statt, denn dort wird vorher der alte Dateiinhalt komplett gelöscht. Bei Binärdateien dagegen bleibt der vorherige Inhalt erhalten und wird je nach Länge des neuen Inhaltes nur teilweise überschrieben.

Hatten Sie in der Datei beispielsweise das Wort TEST stehen und schreiben beim zweiten Anlauf die Buchstabenfolge RO hinein, enthält die Datei danach das Wort ROST.

11.1.3 Random-Access-Dateien

Eine Vorstufe zu den Datenbanken stellen die so genannten Random-Access-Dateien dar. Mit dieser Dateiform können Daten strukturiert abgelegt werden, sodass später auch ein gezielter und wahlfreier Zugriff (engl. Random Access) auf einzelne Teile der Daten möglich ist.

Während Sie in Programmiersprachen wie Pascal oder C++, die eine eigene Typ- und Recorddefinition ermöglichen, komplette Datensätze mit mehreren Feldern definieren können, sind Sie in eMbedded Visual Basic erheblich stärker eingeschränkt. Hier haben Sie nur die Möglichkeit, eine konstante Satzlänge in Bytes vorzugeben.

Man kann sich das der Einfachheit halber wie Zeilen in einer Liste vorstellen. Dabei ist eine konstante Maximallänge für alle Zeilen der Liste definiert. Durch spezielle Befehle können Sie später gezielt auf eine bestimmte Zeile der Liste zugreifen, ohne die restlichen Zeilen vorher lesen zu müssen.

Wichtigste Voraussetzung dafür ist, dass Sie die Satzlänge korrekt berechnen. Ähnlich wie bei den Binärdateien werden auch hier jeweils zwei Bytes für den Typ der Variablen sowie eventuell zwei Bytes für die Stringlänge vorangestellt. Wenn Sie also Zeichenketten mit einer Maximallänge von sechs Zeichen speichern möchten, müssen Sie als Satzlänge 10 (6 + 2 + 2) angeben.

Mit der folgenden Speichern-Routine schreiben Sie drei Namen in eine Random-Access-Datei:

```
Option Explicit

Const Satzlaenge = 6

Private Sub btnSpeichern_Click()
    File1.Open "Testnotiz.rnd", fsModeRandom, , , Satzlaenge + 4
```

```
    ' 2 Byte für Typ + 2 Byte für Stringlänge

    File1.Put "Alfred", 1
    File1.Put "Holger", 2
    File1.Put "Thomas", 3

    File1.Close
End Sub
```

Mit der zu Beginn definierten Konstanten `Satzlaenge` wird die Anzahl der nutzbaren Zeichen pro Datensatz definiert. Beim Öffnen der Datei wird der Modus `fsModeRandom` verwendet, da es sich um eine Random-Access-Datei handelt. Den zweiten und dritten Parameter können Sie hier leer lassen. Als vierter Parameter wird dann die Satzlänge angegeben. Dabei müssen jedoch die vier Bytes für die Zusatzinformation (Variablentyp und Stringlänge) addiert werden.

Anschließend werden über die `Put`-Methode drei Namen in die Datei geschrieben. Der zweite Parameter gibt die Position des Datensatzes an. Zuletzt wird die Datei – wie gehabt – über die `Close`-Methode geschlossen.

Platzieren Sie nun ein ListBox-Objekt mit Namen `lstListe` auf dem Formular. Die TextBox aus dem vorigen Beispiel benötigen wir jetzt nicht mehr. Mit der folgenden Ereignisprozedur werden die gespeicherten Werte nun – in umgekehrter Reihenfolge – in die Listbox eingefügt:

```
Private Sub btnLaden_Click()
    Dim Zeile
    Dim i

    lstListe.Clear

    File1.Open "Testnotiz.rnd", fsModeRandom, , , Satzlaenge + 4
    For i = (File1.LOF / (Satzlaenge + 4)) To 1 Step -1
        File1.Get Zeile, i
        lstListe.AddItem Zeile
    Next

    File1.Close
End Sub
```

Zu Beginn werden zwei Variablen deklariert sowie die ListBox über die Methode `lstListe.Clear` geleert. Anschließend wird die Datei mit exakt derselben Anweisung wie beim Speichern geöffnet. Nun wird eine Schleife durchlaufen, die von der Anzahl der Einträge bis 1 herunterzählt und dabei jedes Mal mit der `Get`-Methode eine Zeile liest und diese über `lstListe.AddItem` zur ListBox hinzufügt.

Die Anzahl der Einträge wird durch eine kleine Formel ermittelt:

```
File1.LOF / (Satzlaenge + 4)
```

Die Methode `LOF` liefert die Größe der Datei in Bytes zurück. Dadurch, dass wir diese nun durch die reale Satzlänge (Länge der Nutzdaten + 4) teilen, erhalten wir die Anzahl der Datensätze.

Achtung: Mögliche Fehlerquelle!

Dieses Verfahren funktioniert leider nur dann einwandfrei, wenn die Maximallänge der Datensätze auch konsequent ausgenutzt wurde. Ansonsten entspricht die reale Länge der Nutzdaten nicht für jeden Datensatz der definierten Konstante `Satzlaenge` und die Formel liefert einen unbrauchbaren Wert. Diesem Problem können Sie begegnen, indem Sie vor dem Speichern jede Zeichenkette bis zur definierten Länge mit Leerzeichen auffüllen.

Das oben genannte Beispiel demonstriert bereits den wesentlichen Vorteil der Random Access-Dateien. Durch Angabe der Position beim Speichern und Lesen kann gezielt auf einen bestimmten Datensatz zugegriffen werden. Dadurch wird z.B. das Auslesen der Datensätze in umgekehrter Reihenfolge erst möglich.

Wenn Sie Random-Access-Dateien geschickt einsetzen, wäre damit sogar eine Adressverwaltung realisierbar. Dazu müssten Sie allerdings immer eine komplette Adresse in eine Zeichenkette packen und diese als Datensatz speichern.

11.2 Eingabe eines Dateinamens über das CommonDialog-Steuerelement

Bisher haben wir der Einfachheit halber mit fest definierten Dateinamen gearbeitet. Das wird natürlich reellen Anforderungen nicht gerecht. Nun wäre es recht einfach, den Dateinamen über eine TextBox abzufragen, doch es geht auch wesentlich komfortabler.

Über das `CommonDialog`-Steuerelement können Sie dem Anwender Dialogfelder präsentieren, durch die dieser Dateinamen zum Speichern eingeben und solche zum Laden auswählen kann.

Als Beispiel hierzu erweitern wir das Programm zum Laden und Speichern von Textdateien. Anstelle der beiden Schaltflächen zum Speichern wird hier eine Schaltfläche mit einer zusätzlichen CheckBox verwendet, über die der Benutzer angeben kann, dass die Daten an eine bestehende Datei angehängt werden.

Wie erwähnt, benötigen wir zusätzlich noch ein `CommonDialog`-Steuerelement, das Sie zunächst über das COMPONENTS-Dialogfeld in das Projekt einbinden müssen und anschließend auf dem Formular platzieren können. Das Dialogfeld erhält dabei automatisch den Namen `CommonDialog1`.

Das Formular sollte nun etwa wie folgt aussehen:

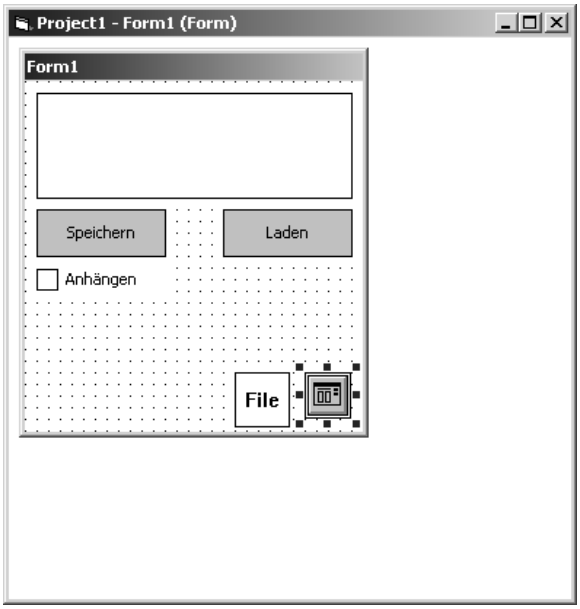

Abb. 11.3 Formular mit CommonDialog-Control

Der Quelltext zu dieser Anwendung ist recht übersichtlich:

```
Option Explicit

Private Sub btnLaden_Click()
    Dim Inhalt As String

    CommonDialog1.ShowOpen

    File1.Open CommonDialog1.FileName, fsModeInput, fsAccessRead
    Inhalt = ""
    While Not File1.EOF
        Inhalt = Inhalt + File1.LineInputString + vbCrLf
    Wend
    txtNotiz.Text = Inhalt
    File1.Close
End Sub

Private Sub btnSpeichern_Click()
    CommonDialog1.ShowSave
```

```
    If chkAnhaengen Then
        File1.Open CommonDialog1.FileName, _
            fsModeAppend, fsAccessWrite
    Else
        File1.Open CommonDialog1.FileName, _
            fsModeOutput, fsAccessWrite
    End If
    File1.LinePrint txtNotiz.Text
    File1.Close
End Sub

Private Sub Form_OKClick()
    App.End
End Sub
```

In der Routine `btnSpeichern_Click` wird zuerst die Methode `CommonDialolg.ShowSave` benutzt, um ein „Datei speichern"-Dialogfeld anzuzeigen. Hier kann der Anwender dann eine Dateibezeichnung eingeben, die in der Objekteigenschaft `CommonDialog1.FileName` abgelegt wird.

Abb. 11.4 Das Dialogfeld Speichern unter

Neben der Eingabe des Dateinamens kann der Anwender auch auswählen, wo die Datei gespeichert wird. Dazu erscheinen in der Auswahlliste ORDNER alle Unterverzeichnisse des Ordners \MY DOCUMENTS, während in der Auswahl ORT alle verfügbaren Speicherorte (HAUPTSPEICHER, Speicherkarte etc.) aufgezählt sind.

Tipp: ORTE zum Speichern von Daten

Über das Auswahlfeld ORT können Sie neben der Voreinstellung HAUPTSPEICHER auch einen anderen Speicherort auswählen. Im Emulator ist hier meist nur die Voreinstellung verfügbar. Auf einem echten Pocket PC sieht das – je nach Gerät – ganz anders aus. Sofern das Gerät über einen Steckplatz für eine Speicherkarte verfügt und eine solche eingelegt ist, erscheint auch diese als Auswahlmöglichkeit.

Bei den aktuellen Geräten der Generation Pocket PC 2002 ist das ROM in einem wiederbeschreibbaren Flash-ROM abgelegt. Der zur Verfügung stehende Platz wird allerdings nicht komplett vom ROM benötigt, sodass auch hier noch etwas Raum übrig ist. In diesem Fall können Sie auch das Flash-ROM als Speicherort auswählen.

Sowohl Speicherkarte als auch Flash-ROM bieten den Vorteil, dass dort gespeicherte Daten selbst bei vollständig entleertem Akku erhalten bleiben.

Doch zurück zum Quelltext:

Nach Eingabe des Dateinamens wird anhand der CheckBox entschieden, ob die Datei zum Schreiben oder zum Anhängen geöffnet wird. Darauf wird der Inhalt der TextBox in die Datei geschrieben und diese geschlossen.

In der Prozedur `btnLaden_Click` sind die Änderungen noch geringer. Hier wird über die Routine `CommonDialog1.ShowOpen` ein Dialogfeld zum Öffnen einer Datei angezeigt:

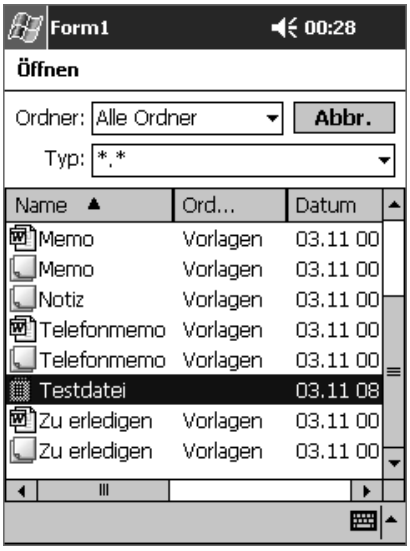

Abb. 11.5 Das Dialogfeld Öffnen

Auch in diesem Dialogfeld ist wieder das Verzeichnis auswählbar. Sobald Sie eine Datei anklicken, wird deren Name in der Objekteigenschaft `CommonDialog1.FileName` abgelegt, die bei der folgenden `Open`-Anweisung verwendet wird.

Der Rest der Prozedur ist Ihnen bereits bekannt: Die Datei wird zeilenweise eingelesen, die Inhalte in die TextBox gesetzt und abschließend die Datei geschlossen.

11.3 Dateioperationen über das FileSystem-Steuerelement

Wie bereits erwähnt, werden durch Einbinden der Komponente FILE SYSTEM CONTROL 3.0 eigentlich zwei Steuerelemente zur Verfügung gestellt. Das FILE-Steuerelement, mit dem Sie Dateien schreiben und lesen können, haben Sie nun kennen gelernt.

Doch wozu genau dient das FILESYSTEM-Steuerelement?

Dieses Steuerelement stellt Funktionen zur Verfügung, um mit Dateien und Verzeichnissen zu arbeiten. Sehen Sie sich einfach mal die verfügbaren Methoden mithilfe des OBJECT BROWSERS an.

Zur Erinnerung: Den Object Browser rufen Sie am schnellsten über die Taste F2 auf.

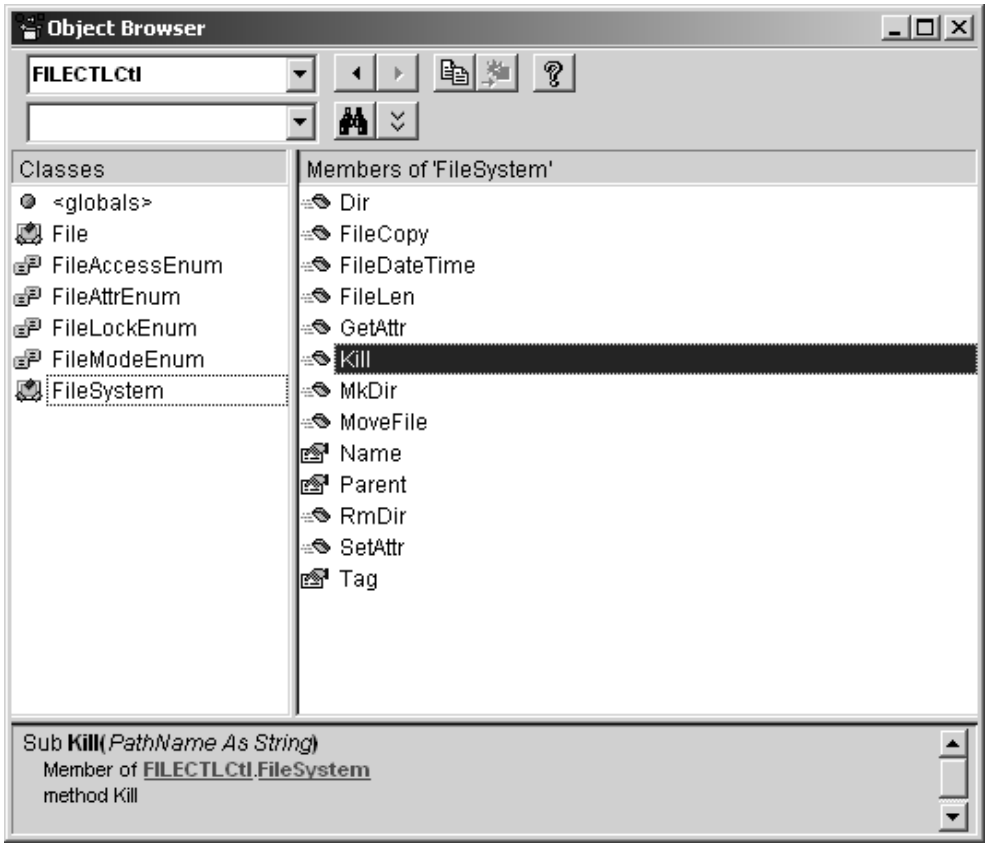

Abb. 11.6 Das FileSystem-Steuerelement im Object Browser

Sie sehen hier verschiedene Funktionen, mit denen Sie Verzeichnisse erstellen (`MkDir`) und löschen (`RmDir`) können. Aber auch solche, mit denen Sie Dateien bewegen (`MoveFile`), kopieren (`FileCopy`), löschen (`Kill`) können oder sogar deren Größe (`FileLen`) ermitteln können.

Ich möchte an dieser Stelle nur ein paar dieser Funktionen detailliert vorstellen. Eine komplette Auflistung finden Sie in *Anhang A*.

11.3.1 Kill zum Löschen von Dateien

Eine der Funktionen, die Sie am ehesten benötigen werden, ist die Funktion `Kill`.

Als Parameter wird lediglich ein Dateiname der Datei erwartet, die anschließend gelöscht wird.

Sinnvoll ist dies beispielsweise dann, wenn Sie mit Random-Access-Dateien arbeiten. In unserem Beispiel zu diesem Thema wurde die Datei geöffnet und es wurden mit der Anweisung `File1.Put` Daten geschrieben. Wenn bereits Daten vorhanden waren, wurden die Datensätze an der übergebenen Position überschrieben, während die anderen erhalten blieben. Dies ist zwar in vielen Fällen sinnvoll, wenn Sie jedoch mit einer leeren Datei starten wollen, sollten Sie die Datei vorher mit der folgenden Anweisung löschen:

```
FileSystem1.Kill "Testnotiz.rnd"
```

Dies wirft jedoch ein anderes Problem auf: Wenn die Datei nicht bereits vorhanden war, erzeugt die Anweisung zum Löschen einen Fehler:

Abb. 11.7 Fehlermeldung bei nicht vorhandener Datei

Daher empfiehlt es sich hier, die Fehlerüberprüfung kurzzeitig auszuschalten. Wenn Sie die Routine zum Speichern wie folgt anpassen, wird die Datei – falls vorhanden – gelöscht, ansonsten erscheint allerdings auch keine Fehlermeldung:

```
Private Sub btnSpeichern_Click()
    On Error Resume Next 'Fehlercheck aus
    FileSystem1.Kill "Testnotiz.rnd"
    On Error GoTo 0 'Fehlercheck an

    File1.Open "Testnotiz.rnd", fsModeRandom, , , Satzlaenge + 4
```

```
    ' 2 Byte für Typ + 2 Byte für Stringlänge

    File1.Put "Alfred", 1
    File1.Put "Holger", 2
    File1.Put "Thomas", 3

    File1.Close
End Sub
```

11.3.2 Dir zum Anzeigen von Verzeichnissen

Mit der Anweisung `Dir` können Sie einen einzelnen Verzeichniseintrag lesen. Dabei wird beim ersten Aufruf durch Parameter definiert, welche Dateien Sie lesen wollen. Bei jedem weiteren Aufruf wird die nächste Datei gelesen, bis alle Verzeichniseinträge des angegebenen Ordners gelesen sind. Dies kann man daran erkennen, dass die Funktion `Dir` dann einen Leerstring zurückgibt.

Hinweis zur Buch-CD:

Die Beispielprojekte zu diesem Thema finden Sie auf der Buch-CD im folgenden Verzeichnis:

`\Beispiele\Kapitel 11\Dateioperationen\`

Erzeugen Sie nun ein neues Projekt und platzieren Sie auf dessen Hauptformular ein LISTBOX-Steuerelement und ein FILESYSTEM-Steuerelement (das Sie vorher natürlich über das COMPONENTS-Dialogfeld ins Projekt einbinden müssen). Wenn Sie anschließend den folgenden Quelltext hinterlegen, wird die ListBox zu Programmbeginn mit den Namen aller Dateien und Verzeichnisse des Hauptverzeichnisses gefüllt:

```
Option Explicit

Private Sub Form_Load()
    Dim strDatei As String

    lstListe.Clear

    strDatei = FileSystem1.Dir("*.*")
    While strDatei > ""
        lstListe.AddItem strDatei
        strDatei = FileSystem1.Dir
    Wend
End Sub
```

```
Private Sub Form_OKClick()
    App.End
End Sub
```

In dieser Anwendung können wir jetzt noch eine Funktion zum Löschen von Dateien unterbringen. Platzieren Sie dazu eine Schaltfläche (`btnLoeschen`) auf dem Formular und hinterlegen Sie dazu folgenden Quelltext:

```
Private Sub btnLoeschen_Click()
    Dim strDatei As String

    strDatei = lstListe.List(lstListe.ListIndex)
    If MsgBox(strDatei & " wirklich löschen ?", vbYesNo) = _
    vbYes Then
        FileSystem1.Kill strDatei
    End If
End Sub
```

Hier wird über die Methode `ListIndex` die Nummer des gewählten Listenelements ermittelt und der Text dazu dann über `lstListe.List` ausgelesen und in die Variable `strDatei` geschrieben. Anschließend erfolgt noch eine kurze Sicherheitsabfrage. Wenn der Anwender diese bejaht, wird die Datei über die `Kill`-Anweisung gelöscht.

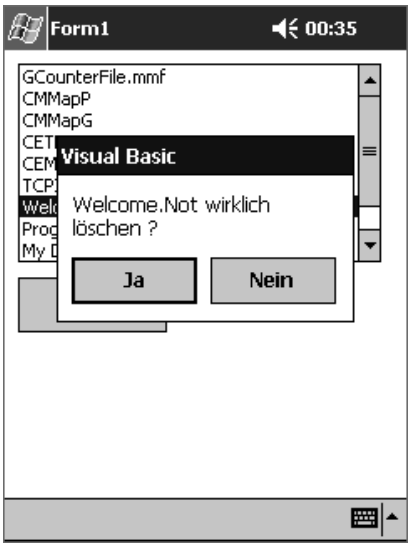

Abb. 11.8 Sicherheitsabfrage vor dem Löschen

Hinweis: Vorsicht beim Löschen von Dateien!

Achten Sie beim Löschen darauf, dass Sie keine wichtigen Systemdateien löschen. Normalerweise erhalten Sie in diesem Fall eine Fehlermeldung ACCESS DENIED, die Sie darauf hinweist, dass die gewählte Datei nicht gelöscht werden kann.

Sollte es Ihnen bei der einen oder anderen Datei dennoch gelingen, diese zu löschen, ist unter Umständen ein Reset des Pocket PCs (bzw. des Emulators) notwendig, um weiter arbeiten zu können.

Dies betrifft aber nur Dateien. In der bisherigen Beispielanwendung werden zwar auch Verzeichnisse mit angezeigt, diese können allerdings nicht mit dem `Kill`-Befehl gelöscht werden.

Nun müssen Sie nur noch dafür sorgen, dass nach dem Löschen einer Datei auch die Liste aktualisiert wird. Dazu könnte man über die Anweisung `Call Form_Load` die vorhin erläuterte Routine aufrufen. Eleganter ist es allerdings, diese in eine neue Prozedur zu verschieben und diese Prozedur dann sowohl von der `Form_Load`-Routine aus als auch nach dem Löschen einer Datei aufzurufen.

Der gesamte Quelltext der Anwendung sieht damit wie folgt aus:

```
Option Explicit

Private Sub VerzeichnisLesen()
    Dim strDatei As String

    lstListe.Clear

    strDatei = FileSystem1.Dir("*.*")
    While strDatei > ""
        lstListe.AddItem strDatei
        strDatei = FileSystem1.Dir
    Wend
End Sub

Private Sub btnLoeschen_Click()
    Dim strDatei As String

    strDatei = lstListe.List(lstListe.ListIndex)
    If MsgBox(strDatei & " wirklich löschen ?", vbYesNo) = _
    vbYes Then
        FileSystem1.Kill strDatei
        Call VerzeichnisLesen
```

```
    End If
End Sub

Private Sub Form_Load()
    Call VerzeichnisLesen
End Sub

Private Sub Form_OKClick()
    App.End
End Sub
```

Nach demselben Verfahren können Sie nun noch Funktionen zum Umbenennen und Kopieren von Dateien etc. hinzufügen und die Anwendung damit zu einem kompletten Dateimanager erweitern.

11.4 Die Registrierdatenbank

Wie zu Beginn des Kapitels bereits angedeutet, eignet sich die Registrierdatenbank vorrangig dazu, kleine Mengen an unstrukturierten Daten abzulegen. In der Praxis wird die Registrierdatenbank meist dazu genutzt, um Konfigurationsparameter für Anwendungen zu hinterlegen.

11.4.1 Arbeiten mit der Registrierdatenbank

Auch auf dem Desktop-PC verfügt jede Windows-Installation (egal, ob Sie mit Windows 95, 98, NT, 2000, ME oder XP arbeiten) über eine Registrierdatenbank.

In dieser Registrierdatenbank (oder kurz Registry) gibt es einzelne Bereiche, in denen Einstellungen zu installierter Hard- und Software sowie zu einzelnen Benutzern gespeichert werden. So schreiben beispielsweise Anwendungen bei deren Installation Konfigurationsparameter (wie z.B. Verzeichnisangaben) in die Registry, die dann bei Ausführung der Anwendung wieder ausgelesen werden.

Wenn Sie die Registrierdatenbank ansehen oder manuell Werte verändern möchten, gibt es dazu den REGISTRIERUNGS-EDITOR (oder kurz REGEDIT). Für dieses Tool gibt es keinen Eintrag im Startmenü, Sie können es aber aufrufen, indem Sie bei geöffnetem Startmenü die Option AUSFÜHREN wählen und anschließend REGEDIT eingeben.

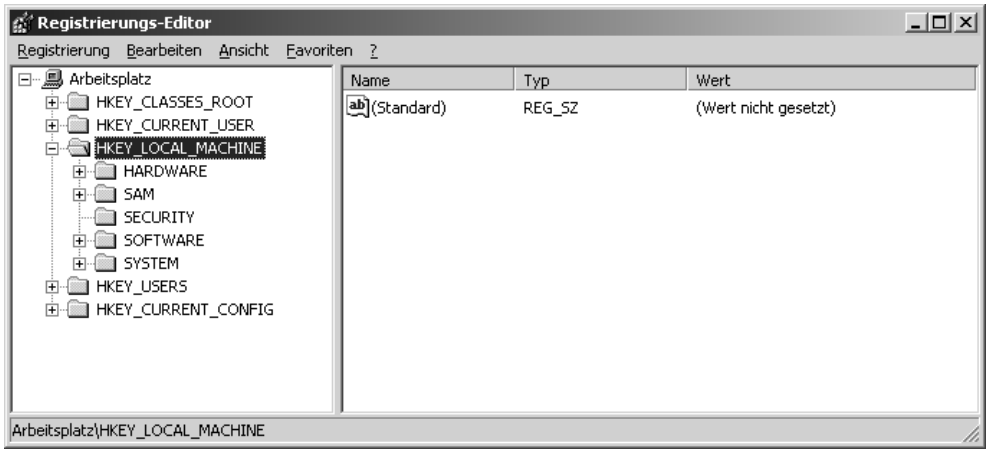

Abb. 11.9 Die Anwendung Regedit auf dem Desktop

Auf der linken Seite sehen Sie die Baumstruktur der Registry, deren Zweige Sie auf- und zuklappen können. Auf der rechten Seite sind die einzelnen Einstellungen des gerade gewählten Zweigs zu sehen. In diesem Bereich können Sie über das Kontextmenü (rechte Maustaste) die Werte einzelner Einträge ändern sowie Einträge löschen und neue hinzufügen.

Wenn Sie nun die Registry Ihres Pocket PCs bearbeiten wollen, funktioniert das sehr ähnlich, nur mit dem Unterschied, dass Sie hier ein anderes Tool benötigen. Für diesen Zweck stellen die eMbedded Visual Tools den WINDOWS CE REMOTE REGISTRY EDITOR zur Verfügung. Dieser wird durch den Menüpunkt TOOLS/REMOTE TOOLS/REGISTRY EDITOR aus der Entwicklungsumgebung aufgerufen. Es erscheint darauf eine Anwendung, die dem Registrierungs-Editor für den Desktop sehr ähnlich ist.

In der Tat können Sie auch mit dem REMOTE REGISTRY EDITOR die Registrierdatenbank des Desktop-PCs bearbeiten (dies ist auch die einzige Registrierdatenbank, die beim ersten Start sichtbar ist). Sie können aber auch über den Menüpunkt CONNECTION/ADD CONNECTION eine Verbindung zu einer Registrierdatenbank auf einem Pocket PC oder dessen Emulator herstellen.

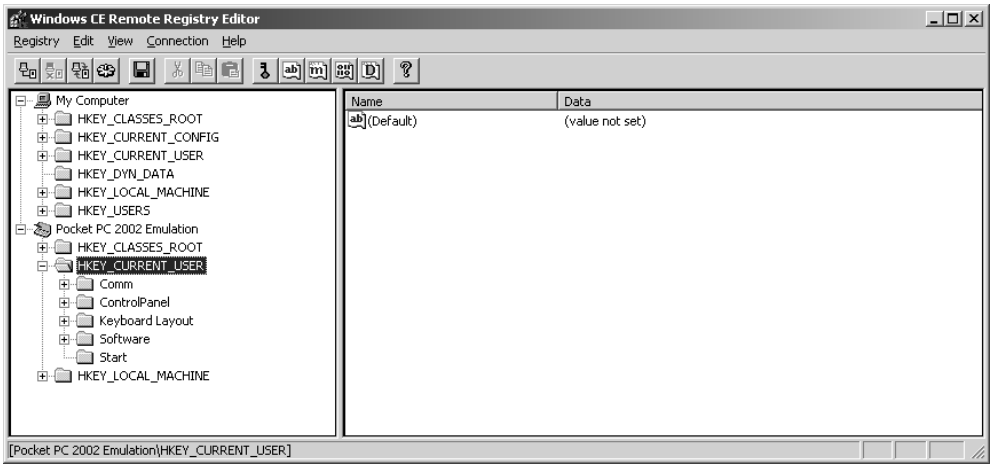

Abb. 11.10 Der Remote Registry Editor

Dadurch erscheint in der Baumstruktur auf der linken Seite ein weiterer Zweig, der die Registry-Einträge des gewählten Gerätes beinhaltet. Wie Sie in der obigen Abbildung sehen, ist die Struktur der Registry auf dem Pocket PC jedoch etwas einfacher als die des Desktop-PCs. Auf dem Pocket PC gibt es nur drei Haupteinträge:

- HKEY_CLASSES_ROOT
- HKEY_CURRENT_USER
- HKEY_LOCAL_MACHINE

HKEY_CLASSES_ROOT enthält systeminterne Daten zu den installierten ActiveX-Komponenten, sodass Sie hier möglichst keine Änderungen vornehmen sollten.

Auf dem Desktop-PC entsprach der Zweig HKEY_CURRENT_USER immer dem Zweig des gerade angemeldeten Users (unterhalb von HKEY_USERS). Auf dem Pocket PC dagegen gibt es nur einen Benutzer. Daher fällt der Zweig HKEY_USERS hier komplett weg und die benutzerspezifischen Daten sind grundsätzlich unterhalb von HKEY_CURRENT_USER gespeichert.

Der dritte „Hauptzweig“ HKEY_LOCAL_MACHINE enthält Einstellungen, die für dieses Gerät gelten. Da beim Pocket PC aber nicht zwischen verschiedenen Benutzern unterschieden wird, macht es eigentlich auch keinen Unterschied, ob Sie Ihre Einstellungen im Zweig HKEY_LOCAL_MACHINE oder im Zweig HKEY_CURRENT_USER speichern.

Wichtiger ist, dass Sie anwendungsspezifische Einstellungen unterhalb des Zweigs SOFTWARE ablegen (den es sowohl unterhalb von HKEY_LOCAL_MACHINE als auch unterhalb von HKEY_CURRENT_USER gibt).

Hinweis: Registry und der Pocket PC-Emulator

Mit dem Pocket-PC-Emulator kommt es manchmal zu Problemen, wenn Sie versuchen, Einträge in die Registry zu schreiben oder diese zu ändern. Wenn Sie also probeweise einen Eintrag manuell hinzufügen wollen, müssen Sie dies auf einem echten Pocket PC tun.

Dasselbe gilt für die programmgesteuerten Änderungen der Registry, die im nächsten Abschnitt folgen.

11.4.2 Lesen und Schreiben von Werten

Unter Visual Basic für den Desktop PC gibt es bereits fertige Funktionen (`GetSetting` und `SaveSetting`, `DeleteSetting`) zum Lesen, Schreiben und Löschen von Registry-Einträgen. eMbedded Visual Basic verfügt leider über keine vergleichbaren Funktionen, sodass man diese über API-Aufrufe nachbilden muss.

Hinweis zur Buch-CD:

Das Beispielprojekt zu diesem Thema finden Sie auf der Buch-CD im folgenden Verzeichnis:

`\Beispiele\Kapitel 11\Registry\`

Den kompletten Source der Registry-Funktionen finden Sie auf der Buch-CD in der Datei MODREGISTRY.BAS. Auf den Quelltext der drei Funktionen detailliert einzugehen, würde an dieser Stelle zu weit führen, daher erkläre ich nur kurz deren Funktionsweise anhand der jeweiligen Prozedur- bzw. Funktions-Deklaration.

Alle drei Routinen erhalten als Parameter den Namen der Applikation, der Sektion sowie des Schlüssels. Dabei sind die Applikation und die Sektion Ebenen in der Baumhierarchie der Registrierdatenbank, während der Schlüssel ein Wert innerhalb der Sektion ist.

Die Funktion `GetSettingCE` ermittelt den Wert des durch die oben genannten Parameter definierten Schlüssels, der als Funktionswert zurückgegeben wird.

```
Public Function GetSettingCE(ByVal AppName As String, _
                             ByVal Section As String, _
                             ByVal Key As String) As Variant
```

Die Funktion `SaveSettingCE` schreibt einen Wert, der als vierter Parameter übergeben wird, in den Schlüssel, der durch die restlichen drei Parameter definiert wird.

```
Public Sub SaveSettingCE(ByVal AppName As String, _
                         ByVal Section As String, _
```

```
                              ByVal Key As String, _
                              ByVal Setting As String)
```

Die Funktion `DeleteSettingCE` schließlich löscht einen Schlüssel aus der Registrierdatenbank. Wenn Sie den Parameter für den Schlüssel leer lassen, wird sogar die ganze Sektion aus der Registry entfernt.

```
Public Sub DeleteSettingCE(ByVal AppName As String, _
                           ByVal Section As String, _
                           ByVal Key As String)
```

Damit die Bedeutung von Applikation, Sektion und Schlüssel klarer wird, wollen wir ein Beispiel entwerfen, das diese Funktionen benutzt. Erzeugen Sie dazu ein leeres Projekt und fügen Sie diesem das Basic-Modul hinzu, durch das die genannten Routinen definiert werden (MODREGISTRY.BAS).

Auf dem Hauptformular erzeugen Sie nun vier TextBoxen (`txtApplication`, `txtSection`, `txtKey` und `txtSetting`) mit dazu passenden Labels sowie drei Schaltflächen (`btnSchreiben`, `btnLesen` und `btnLoeschen`). Das Formular sieht nun etwa so aus:

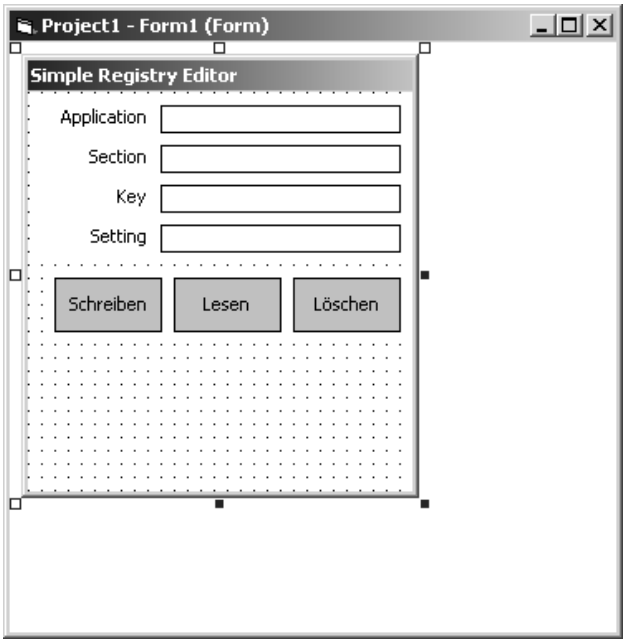

Abb. 11.11 Die Beispielanwendung im Entwurfsmodus

Idee dieser Anwendung ist, dass der Benutzer durch die Schaltflächen Werte, die durch die TextBoxen definiert werden, in die Registry schreiben, aus ihr lesen oder löschen kann.

Der Quelltext zu dieser Anwendung ist – abgesehen von der eingebundenen Moduldatei – sehr übersichtlich:

```
Option Explicit

Private Sub btnLesen_Click()
    txtSetting.Text = GetSettingCE(txtApplication.Text, _
        txtSection.Text, txtKey.Text)
End Sub

Private Sub btnLoeschen_Click()
    DeleteSettingCE txtApplication.Text, txtSection.Text, _
        txtKey.Text
End Sub

Private Sub btnSchreiben_Click()
    SaveSettingCE txtApplication.Text, txtSection.Text, _
        txtKey.Text, txtSetting.Text
End Sub

Private Sub Form_OKClick()
    App.End
End Sub
```

Durch einen Klick auf die Schaltfläche LESEN wird über die Funktion `GetSettingCE` der durch die ersten drei Eingabefelder definierte Wert aus der Registry gelesen und in dem vierten Feld (`txtSetting`) angezeigt.

Wählt der Anwender die Schaltfläche `btnLoeschen`, wird der Schlüssel oder die Sektion, die durch die ersten drei Felder definiert sind, aus der Registrierdatenbank gelöscht.

Über die Schaltfläche `btnSchreiben` wird der Wert aus dem vierten Feld in den Schlüssel geschrieben, der durch die anderen drei Felder beschrieben wird.

Experimentieren Sie mit dieser Anwendung ruhig ein wenig herum und betrachten Sie die Auswirkungen im REMOTE REGISTRY EDITOR. Dann sollte die Funktionsweise von Registrierdatenbank sowie der verwendeten Routinen schnell klar werden.

Hinweis: Bereich der Registry ändern

Die Einträge, die mithilfe der drei Routinen bearbeitet werden, beziehen sich alle auf den Zweig `HKEY_CURRENT_USER`. Sie können dies aber ändern, indem Sie die Zeile ...

```
RegistryRoot = HKEY_CURRENT_USER
```

... in der Datei MODREGISTRY.BAS entsprechend anpassen und die Konstante `HKEY_CURRENT_USER` durch eine der beiden anderen Varianten (`HKEY_LOCAL_MACHINE` oder `HKEY_CLASSES_ROOT`) ersetzen.

Für die meisten Fälle dürfte HKEY_CURRENT_USER aber die bessere Wahl sein, zumal dieser Bereich am wenigsten mit vordefinierten Systemeinstellungen überladen ist.

11.5 Zusammenfassung

Sie haben in diesem Kapitel verschiedene Möglichkeiten kennen gelernt, um Daten dauerhaft auf dem Pocket PC zu speichern. Dabei wurden anfangs die verschiedenen Dateiformen (Textdatei, Binärdatei und Random-Access-Datei) besprochen.

Anschließend wurde aufgezeigt, wie Sie über vordefinierte Dialogfelder Dateinamen vom Anwender erfragen können und wie Sie mit dem Dateisystem arbeiten (Verzeichniseinträge anzeigen, Dateien löschen etc.).

Schließlich haben Sie die Funktionsweise der Registrierdatenbank kennen gelernt und ein paar fertige Routinen gezeigt bekommen, mit denen Sie diese aus eigenen Anwendungen heraus bearbeiten können.

Somit kennen Sie nun die wichtigsten Methoden, um unstrukturierte Daten (in Dateien) oder kleine Mengen von nicht einheitlich strukturierten Daten (in der Registrierdatenbank) zu verwalten.

Was Ihnen noch fehlt, ist eine Möglichkeit, größere Datenmengen strukturiert zu speichern. Dies ist Thema der nächsten Kapitel, die sich mit Datenbanken beschäftigen.