

## KAPITEL 4

### KONTROLLSTRUKTUREN

#### 4.1. Wiederholungen

Inzwischen ist Ihnen sicher klar geworden, daß Computer auf eine geradezu ermüdende Weise stupid sind, man muß (leider) sehr präzise bei der Formulierung jedes Auftrags sein. Ein Vorteil ist Ihnen aber sicher ebenfalls klar geworden, die Maschine ist bei der Ausführung der Anweisungen schnell. Diese zweite Eigenschaft läßt sich wunderbar ausnutzen, die erste wettzumachen. Vieles von dem, was ein Computer durchführt, besteht nämlich in der Wiederholung einer bestimmten Anweisung. Stellen Sie sich vor, Sie sollten die Zahlen von 1 bis 100 oder gar 1000 aufschreiben, Sie wären vermutlich entsetzt und gleichzeitig gelangweilt - und würden schon deshalb mit großer Wahrscheinlichkeit Fehler machen. Gerade solche Wiederholungen aber stellen für den Computer die wenigste Mühe dar, dann jedenfalls, wenn Sie ihn durch passende Anweisungen, die Sie im Folgenden kennenlernen, dazu aufgefordert haben.

#### Die REPEAT - Schleife

Component Pascal stellt für Wiederholungen vier verschiedene Möglichkeiten zur Verfügung, die Sie alle in der Datei Schleife.odc finden. Der Deklarationsteil der Prozedur *Start* macht Ihnen höchstwahrscheinlich keine Probleme, dort werden zwei Konstanten und drei Variablen deklariert. Auch in den ersten beiden Zeilen des Anweisungsteils finden Sie nur Bekanntes. Neu ist dagegen der Anweisungsblock, der durch die Component Pascal Schlüsselwörter *REPEAT ... UNTIL ...* geklammert wird. Die einzige Klammer, die Sie bisher kennengelernt haben, lautet *BEGIN ... END* und faßt alles zusammen, was in den Anweisungsteil einer Kommando-prozedur gehört, wobei die innerhalb des Blocks stehenden Anweisungen genau einmal der Reihe nach abgearbeitet werden.

Anders dagegen der *REPEAT ... UNTIL*-Block. Der Compiler "merkt" sich beim ersten Erreichen des Schlüsselworts *REPEAT* dessen Position. Die zwischen *REPEAT* und dem Schlüsselwort *UNTIL* stehende Anweisungsfolge wird dann wie andere Anweisungsfolgen der Reihe nach abgearbeitet. Wenn das Programm das zweite Schlüsselwort *UNTIL* erreicht, wird die Bedingung geprüft, die im Anschluss an das *UNTIL* steht. Im Fall der vorliegenden *REPEAT*-Schleife heißt diese Bedingung *Index = 5* und ist - allgemein gesprochen - ein Boole'scher Ausdruck, der falsch oder wahr sein kann. Sofern das Ergebnis der Prü-

fung *FALSE* ist, wird die innerhalb der *REPEAT*-Schleife stehende Anweisungsfolge solange erneut durchlaufen, bis die Prüfung *TRUE* ergibt, danach wird mit der anschließenden Anweisung fortgefahren.

Wichtig bei dieser Schleifenform ist, daß Sie als die Programmierenden selbst dafür sorgen müssen, wie die beendende Bedingung erreicht, also *TRUE* wird. Im vorliegenden Fall geschieht das dadurch, daß zu Beginn jedes Schleifendurchlaufs der Wert der Variablen *Index* erhöht wird. Dabei muß natürlich vor Eintritt in die Schleife diese Variable initialisiert, d. h. ihr muß ein Wert zugewiesen worden sein, und selbstverständlich muß dieser kleiner als der beendende Wert oder ihm zumindest gleich sein, denn andernfalls würde die Schleife bei dem gegebenen Abbruchkriterium Gleichheit ( $Index = 5$ ) nie beendet werden können. Endlosschleifen, die auf solche Fehler zurückzuführen sind oder darauf, daß vergessen wurde, die Kontrollvariable beim erneuten Schleifendurchgang zu verändern, sind "beliebte" Programmierfehler, die ärgerlicherweise auch erst bei der Ausführung des Programms in Erscheinung treten, da der Compiler naturgemäß logische Fehler dieser Art nicht erkennen kann.

Etwas anderes wäre es, wenn die zur Beendigung der Schleife führende Zeile *UNTIL Index >= 5* hieße. Selbst wenn Sie die Variable mit  $Index := 6$  initialisiert hätten, würde die Schleife zwar einmal durchlaufen, ein weiterer Durchgang fände jedoch nicht statt, da die Abbruchbedingung  $Index >= 5$  erfüllt wäre. Sie sehen, es hängt von Ihrer Programmierarbeit ab, was geschieht; die Wahl eines geeigneten, möglichst fehlersicheren Abbruchkriteriums liegt in Ihrer Verantwortung.

#### Die WHILE - Schleife

Als nächste Schleife finden Sie die *WHILE*-Schleife, die in diesem Programm genau das gleiche Ergebnis erzeugt wie ihre Vorgängerin. Die *WHILE*-Schleife ist gegeben durch die Component Pascal Schlüsselwörter *WHILE ... DO ... END*, wobei statt der Punkte zwischen *WHILE* und *DO* wiederum jeder beliebige Boole'sche Ausdruck stehen kann, während hinter dem *DO* der zur Schleife gehörende Anweisungsblock steht. Der Unterschied zur *REPEAT ... UNTIL*-Schleife besteht darin, daß in der *WHILE ... DO ... END*-Schleife sofort nach Erreichen des Schlüsselworts *WHILE* der dort stehende Boole'sche Ausdruck geprüft wird. Ist das Ergebnis der Prüfung *FALSE*, wird das Programm mit der ersten Anweisung hinter dem *END* der Schleife fortgesetzt, ist das Ergebnis dagegen *TRUE*, wird der Schleifeninhalt abgearbeitet und beim Erreichen des *END* erfolgt ein Sprung zurück zum Anfang *WHILE*, wo die Bedingung erneut geprüft wird. Je nach Ausgang dieser Prüfung wird der Schleifeninhalt wiederum durchlaufen oder die Schleife mit einem Sprung zur Anweisung hinter dem *END* der Schleife verlassen. Ebenso wie bei der *REPEAT*-Schleife müssen Sie bei der *WHILE*-Schleife selbst dafür sorgen, daß das gewählte Abbruchkriterium erreicht, die dem *WHILE* folgende Bedingung *FALSE* wird.

Ein inhaltlich wesentlicher Unterschied zwischen der *REPEAT*- und der *WHILE*-Schleife zeigt sich darin, wann die zum Schleifenabbruch führende Bedingung geprüft wird und welchen Wert sie für die Beendigung der Schleifenbearbeitung haben muß. Bei der *WHILE*-Schleife wird die Bedingung bei Eintritt in die Schleife geprüft und diese wird solange erneut durchlaufen, wie die Bedingung *TRUE* ist, während bei der *REPEAT*-Schleife die Prüfung am Ende erfolgt und die Schleife solange durchlaufen wird, wie die Abbruchbedingung *FALSE* ist. Hätte bei der *WHILE*-Schleife die Initialisierung geheißen:  $Index := 5$ , gäbe es keinen einzigen Schleifendurchlauf, das Programm würde mit der folgenden Anweisung fortgesetzt. Dagegen wird eine *REPEAT*-Schleife auf alle Fälle wenigstens ein Mal durchlaufen.

Dieser Unterschied führt zu schwerwiegenden Konsequenzen, wenn innerhalb einer Schleife eine kritische Bedingung auftritt. Beispielsweise könnte innerhalb der Schleife durch *Index* dividiert werden, was dazu führte, daß durch Null dividiert würde, sofern *Index* aus irgendeinem Grund den Wert Null enthielte. Bekanntermaßen ist die Division durch Null mathematisch nicht möglich, im Computer führt diese Operation zu einem sogenannten Laufzeitfehler, bei dem die Programmausführung in der Regel vom Rechner abgebrochen wird. Während Sie aber mit der *WHILE*-Schleife die Möglichkeit haben, diesen Fall durch die Konstruktion *WHILE Index # 0 DO ... END* auszuschließen, also den Eintritt in die Schleife zu vermeiden, würde die *REPEAT*-Schleife wenigstens einmal durchlaufen werden, da bei dieser Schleife die Abbruchbedingung erst im Anschluss geprüft wird. Man nennt die *REPEAT*-Schleife deshalb auch nachprüfende Schleife, während die *WHILE*-Schleife als vorprüfende Schleife bezeichnet wird.

#### Die FOR - Schleife

Die dritte Möglichkeit, Wiederholungen durchführen zu lassen, stellt die *FOR*-Schleife dar, die durch die Component Pascal Schlüsselwörter *FOR ... TO ... BY ... DO ... END* gebildet wird. Die Schleifendefinition enthält statt der Punkte zwischen *FOR ... TO* die sogenannte Kontrollvariable (auch Zähl- oder Laufvariable genannt) mitsamt der Zuweisung eines Startwertes, zwischen den Wörtern *TO ... BY* steht der Endwert, zwischen *BY ... DO* eine Konstante, mit der die Schrittweite für die Wiederholung angegeben wird. Die Kontrollvariable muß vom Typ *INTEGER* sein, Start- und Endwert sowie Schrittweite dürfen folglich nur Ganzzahlwerte annehmen. Zwischen *DO ... END* stehen die auszuführenden Anweisungen. Das Schlüsselwort *BY* und die Angabe einer Schrittweite sind fakultativ, sie können also weggelassen werden, in diesem Fall setzt der Compiler für die Schrittweite automatisch den Wert "+1" ein, andernfalls gilt der dem Schlüsselwort *BY* folgende Wert. Sie finden beide Varianten der *FOR*-Schleife im Programm.

*FOR*-Schleifen bieten den Vorteil, daß die Kontrollvariablen automatisch geändert werden, man muß sich um die beendende Bedingung also keine besonderen Gedanken machen. Der Abarbeitungsmechanismus einer *FOR*-Schleife ist dafür ein wenig komplizierter als in den beiden vorigen Fällen. Im Programm-

ablauf müssen nämlich rechnerintern außer der Rücksprungmarke am *FOR* der jeweilige Wert der Kontrollvariablen, der Endwert und die Schrittweite zwischengespeichert werden. Diese Werte werden beim ersten Eintritt in die *FOR*-Schleife erfaßt und danach wird geprüft, ob der Startwert größer (bzw. kleiner bei negativer Schrittweite) als der Endwert ist. Solange die Prüfung *FALSE* ergibt, wird der Schleifeninhalt abgearbeitet und am Ende des zur Schleife gehörenden Anweisungsblocks erfolgt ein Rücksprung zum *FOR* des Schleifenanfangs. Dort wird der Wert der Kontrollvariablen um die Schrittweite geändert, erneut geprüft und die Schleife entweder ein weiteres Mal durchlaufen oder mit der ersten nachfolgenden Anweisung fortgefahren.

In komplizierteren Programmen ist es natürlich möglich, die Zahl der Schleifendurchgänge von Werten abhängig zu machen, die von bestimmten, vor Eintritt in die Schleife geprüften Bedingungen festgelegt worden sein können. Achten Sie in einem solchen Fall bitte darauf, daß sich bei der Bestimmung dieser Werte eine ganze Zahl ergeben muß, da die Kontrollvariable einer *FOR*-Schleife nur Werte vom Typ *INTEGER* akzeptiert.

Auf zwei besondere Fälle möchte ich noch hinweisen. Bei einer programmabhängigen Festlegung von Anfangs- und Endwerten für die Kontrollvariable könnte es geschehen, daß der Anfangswert höher (niedriger bei einer abwärts zählenden *FOR*-Schleife) als der Endwert ist. In diesem Fall würde die Schleife überhaupt nicht durchlaufen, die Programmabarbeitung setzte mit der ersten auf die Schleife folgenden Anweisung fort. Dagegen würde bei einer ebenfalls möglichen Anweisung *FOR Index := 5 TO 5 DO ... END* der zwischen *DO* und *END* stehende Anweisungsblock genau ein Mal durchlaufen, weil die Kontrollvariable erst nach dem Rücksprung an den Schleifenanfang geändert und anschließend geprüft wird, ob der Endwert überschritten (bzw. unterschritten) wurde.

Beachten Sie bitte auch, daß die Kontrollvariable einer *FOR*-Schleife durch keine Anweisung innerhalb der Schleife verändert werden sollte. Die anfangs festgelegten Bedingungen sollten während aller Schleifendurchläufe konstant bleiben (wenn Sie erfahren wollen, was im gegenteiligen Fall geschehen kann, entfernen Sie in der zweiten *FOR*-Schleife die Kommentarklammern um die Anweisung *DEC(Index)*, kompilieren Sie das Programm erneut und lassen Sie es anschließend ausführen).

#### Die LOOP - Schleife

Die letzte der in Component Pascal existierenden Wiederholungsanweisungen stellt die *LOOP*-Schleife dar. Sie ist gegeben durch die Schlüsselwörter *LOOP ... END*, zwischen denen alle zu wiederholenden Anweisungen stehen. Diese Schleifenart enthält weder sich automatisch ändernde Kontrollvariablen wie die *FOR*-Schleife, noch gibt es wie bei der *WHILE*- oder der *REPEAT*-Schleife am Anfang oder am Ende eine Bedingung, die zur Beendigung der Wiederholungen führte. Die *LOOP*-Schleife wird in dem

Moment beendet, in dem das Programm bei der Abarbeitung der Schleifenanweisungen auf das Component Pascal Schlüsselwort *EXIT* stößt.

Nach Ihren bisherigen Kenntnissen müßten Sie jetzt stutzen und sich sagen, daß dieses Wort irgendwo innerhalb, spätestens als letztes in der Schleife stehen muß, die Schleife also bei linearer Abarbeitung der Schleifenanweisungen nur teilweise oder bestenfalls einmal ganz durchlaufen würde. Damit hätten Sie nach dem Stand Ihres Wissens zweifellos recht, es muß also eine Ihnen bisher unbekannte Möglichkeit geben, das Wort *EXIT* zu "verstecken", um mehr als einen Schleifendurchlauf möglich zu machen. Diese Möglichkeit gibt es in Form einer sogenannten "bedingten Verzweigung". Damit ist gemeint, daß an einer bestimmten Stelle des Programms in Abhängigkeit von einer Bedingung ein Teil der Schleifenanweisungen ausgeführt oder ignoriert wird.

Eine solche Bedingung wird in Component Pascal durch die Schlüsselwörter *IF ... THEN ... END* realisiert, wobei zwischen *IF ... THEN* ein beliebiger Boole'scher Ausdruck und zwischen *THEN ... END* die bei Zutreffen dieser Bedingung abzuarbeitenden Anweisungen stehen (Genauer erfahren Sie im Zusammenhang mit dem Programm *IfDemo.odc* im folgenden Abschnitt).

In der *LOOP*-Schleife des Moduls *TutSchleife* finden Sie die Bedingung für die Beendigung der Schleifenwiederholungen in den Zeilen *IF Index = 25 THEN ... EXIT END*. Die Anweisungen zwischen *THEN ... EXIT* werden nur ausgeführt, wenn die Bedingung *Index = 25* erfüllt ist. Da *Index* bei Eintritt in die *LOOP*-Schleife mit Null initialisiert und bei jedem Schleifendurchlauf um Eins inkrementiert wird, gibt es 26 Schleifendurchläufe, bevor das beendende *EXIT* erreicht werden kann, es wird das gesamte Alphabet im Log-Fenster ausgegeben.

#### 4.2. Bedingte Verzweigungen

Mit dem Programm *IfDemo.odc* lernen Sie genaueres über die im vorigen Abschnitt bei der *LOOP*-Schleife beschriebenen Verfahren, sich in einem Programm für eine von zwei (oder auch mehr, wie Sie sehen werden) Möglichkeiten einer bedingten Verzweigung zu entscheiden.

Die Wichtigkeit von Verzweigungen liegt darin, daß die Entscheidung, welcher der Zweige gewählt wird, auch während des Programmablaufs getroffen werden kann, abhängig davon, welche Bedingungen aktuell gelten. Beispielsweise wird bei automatischen Heizungsregelungen das Ergebnis einer externen Temperaturmessung das Programm veranlassen, die Steuerkurve der Heizung entsprechend dem Meßergebnis zu verstellen. Die Fähigkeit zu derartigen Entscheidungen macht bedingte Verzweigungen zu äußerst nützlichen Programmierwerkzeugen.

Der vollständige Component Pascal Ausdruck für die Verzweigung besteht aus den Schlüsselwörtern *IF ... THEN ... ELSIF ... THEN ... ELSE ... END*, wobei die *ELSIF*- und *ELSE*-Teile fehlen dürfen. Die verschiedenen Varianten finden Sie im Programm IfDemo.odc.

Der Anweisungsteil des Programms besteht aus einer umfassenden *FOR*-Schleife, die nichts weiter tut, als die Variable *Index* nacheinander auf die Werte 1 bis 8 zu setzen und diese jeweils auszugeben. Die erste der in diesem Abschnitt interessierenden *IF*-Abfrage innerhalb der *FOR*-Schleife fragt bei jedem Durchlauf nach dem Wert von *Index*. Falls dieser kleiner als 4 ist, wird ein entsprechender Text ausgegeben, im anderen Fall geschieht nichts. Es gibt keine sonstigen Teile in dieser *IF*-Abfrage, man spricht deshalb auch von einer einseitigen Verzweigung.

Die nächste *IF*-Abfrage enthält einen der eben "vermißten" sonstigen Teile, den *ELSE*-Teil, es handelt sich um eine zweiseitige Verzweigung. Allgemein steht bei dieser Form der Entscheidungsanweisungen zwischen den Schlüsselwörtern *IF* und *THEN* ein Boole'scher Ausdruck, der bei Eintritt in die Verzweigung ausgewertet wird. Wenn das Ergebnis *TRUE* ist, wird der auf *THEN* folgende Anweisungsblock ausgeführt, sonst erfolgt ein Sprung zum *ELSE*-Teil - falls dieser existiert - und der dort stehende Anweisungsblock wird ausgeführt, anderenfalls erfolgt ein Sprung zum Ende der *IF*-Abfrage.

Die dritte *IF*-Abfrage des Programms ist ebenfalls eine zweiseitige Verzweigung, sie enthält einen *THEN* und einen *ELSE*-Teil. Dieser enthält eine weitere *IF*-Abfrage, die ihrerseits aus einem *THEN*- und einem *ELSE*-Teil besteht, wobei der *THEN*-Teil wiederum eine Verzweigung enthält, die jedoch mit den Schlüsselwörtern *ELSIF ... THEN ...* eingeleitet wird. Der Unterschied zwischen einem *ELSE*-Teil und einem *ELSIF*-Teil besteht darin, daß der *ELSE*-Teil nur ein Mal und nur als letzter Teil in der Verzweigung vorkommen darf, aber nicht notwendig vorkommen muß, wie Sie an der ersten *IF*-Abfrage bereits gesehen haben. Dagegen können beliebig viele *ELSIF*-Teile existieren, die alle Bestandteile derselben *IF*-Abfrage sind, weshalb sie nur ein *THEN* nach einer weiteren Boole'schen Bedingung, aber kein eigenes *END* benötigen.

Die letzte *IF*-Abfrage ist wieder einfacher, sie enthält in ihrem *THEN*-Teil eine weitere einseitige *IF*-Abfrage; das Verständnis dieser Konstruktion wird Ihnen vermutlich keine Schwierigkeiten bereiten.

Es ist offensichtlich, daß der Anzahl der *ELSIF*-Teile ebensowenig prinzipielle Grenzen gesetzt sind wie der Anzahl weiterer innerhalb der *THEN*- oder *ELSE*-Teile einer Verzweigung stehender *IF*-Abfragen. Beachten Sie aber, daß bei allen Verzweigungen während der Abarbeitung stets nur genau einer der Zweige durchlaufen wird. Dies bedeutet, daß Sie dafür sorgen müssen, bei den Entscheidungskriterien Überschneidungen strikt zu vermeiden. Es wäre also wegen der eben erwähnten Eindeutigkeit der Zweigauswahl sinnlos, die im *ELSE*-Teil der dritten, geschachtelten *IF*-Abfrage auftauchende weitere *IF*-Bedingung etwa in der Form *IF Index = 2 THEN ...* zu schreiben, da dieser Fall innerhalb des fraglichen *ELSE*-Zweiges nicht auftreten kann. Die Bemerkungen "7 ist nicht 8", "! Ooh Zahlen !" oder "! Ach Zahlen !"

können für den *Index*-Wert 2 nicht erscheinen, denn der gesamte *ELSE*-Teil wird für den *Index*-Wert 2 nicht durchlaufen, das Programm setzt mit der ersten nachfolgenden Anweisung (*Out.Ln*) fort.

Diese Tatsache wird für Sie von Bedeutung sein, wenn Sie darangehen müssen, noch mehr als die hier gezeigten *IF*-Abfragen in einander zu schachteln, aber auch dann, wenn es um Mehrfachverzweigungen geht (s. dazu das Programm CaseDemo.odc, Programm 4 dieses Kapitels), denn es ist klar, daß die dabei möglichen logischen Fehler nicht vom Compiler entdeckt werden können, sie werden sich erst bei der Programmausführung bemerkbar machen.

Sie sehen vielleicht jetzt schon, wie wichtig es ist, ein fertiges Programm so zu testen, daß derartige logische Fehler mit Sicherheit entdeckt werden; leider geschieht das selbst bei professionellen Programmen nicht immer, "seltsame Ereignisse" oder "Programm-Abstürze" beruhen häufig darauf, daß die Konstruktion der Tests unsauber war und die Inkonsistenzen des Programms deshalb nicht rechtzeitig bemerkt worden sind.

Werten Sie bitte das Programm "zu Fuß" aus und kontrollieren Sie, ob Ihre Auswertung mit der Programmausgabe übereinstimmt, es ist wesentlich für Ihre künftige Arbeit, daß Sie die hier vorgestellten Verzweigungen vollständig begriffen haben.

### 4.3. Schleifen - Zweige - Schleifen

Das 3. Programm dieses Kapitels - WhileIf.odc - erfordert keinen umfangreichen Kommentar, es bringt keine Neuigkeiten, vielmehr verbindet es die in diesem Kapitel bisher vorgestellten Kontrollstrukturen - Schleifen und Verzweigungen - so, daß Sie in übersichtlicher Form sehen können, wie diese Konstrukte miteinander verknüpfbar sind. Es ist vom Aufbau her einfach gegliedert, eine umfassende *WHILE*-Schleife enthält eine *IF*-Abfrage mit *THEN*- und *ELSE*-Teil, wobei der *THEN*-Teil seinerseits eine *FOR*-Schleife, der *ELSE*-Teil eine *REPEAT*-Schleife enthält.

Sie sollten auch dieses Programm im "Handdurchlauf" auswerten und sich die (von Ihnen erarbeitete) Bildschirmausgabe aufschreiben, bevor Sie das Programm vom Rechner ausführen lassen; ich wiederhole mich, es ist äußerst wichtig für das weitere Verständnis der Programmiersprache Component Pascal, daß Sie die hier vorgestellten Syntax-Elemente beherrschen. Falls Ihre Notizen nicht mit der vom Computer erzeugten Ausgabe übereinstimmen, sollten Sie sich unbedingt noch einmal die Programme Schleife.odc und IfDemo.odc ansehen, die die Grundlagen für das vorliegende Programm WhileIf.odc darstellen.

#### 4.4. Mehrfachverzweigungen

Mit dem Programm CaseDemo.odc lernen Sie eine Erweiterung der Möglichkeiten kennen, die durch die IF-Abfrage gegeben sind, die CASE-Abfrage. Die CASE-Abfrage erlaubt Ihnen die Entscheidung zwischen (im Prinzip) beliebig vielen Möglichkeiten, wobei natürlich ebenfalls stets nur genau eine der Möglichkeiten wählbar sein darf, die Entscheidungen müssen also auch hier eindeutig sein.

Die vollständige Abfrage besteht aus den Schlüsselwörtern *CASE ... OF ... ELSE ... END*, wobei der *ELSE*-Teil fehlen darf. Die Verwendung der CASE-Abfrage entnehmen Sie am einfachsten dem Beispiel des Programms. Im Anschluß an das Schlüsselwort *CASE* steht eine einfache Variable, die vom Typ *CHAR* oder *INTEGER* sein muß, gefolgt vom Schlüsselwort *OF*. Diesem folgt die Aufzählung der verschiedenen möglichen Fälle, das sind im Fall der gegebenen *INTEGER*-Variablen *Kind* ganze Zahlen. Die Fälle können, wie Sie sehen, einzeln aufgeführt sein oder auch aneinandergereiht, in diesem Fall müssen sie durch Kommata voneinander getrennt werden. Mehrere Werte eines zusammenhängenden Teilbereichs des jeweiligen Typs (hier: *INTEGER*) können durch einen Anfangs- und einen Endwert, die durch (genau) zwei Punkte voneinander getrennt sind, angegeben werden. Jede einzelne Fallangabe wird durch einen Doppelpunkt von dem Anweisungsblock getrennt, der bei Eintritt des entsprechenden Falles ausgeführt werden soll. Die einzelnen Fälle müssen durch einen vertikalen Separator (Latin1-Zeichen 124) von einander getrennt werden, den Sie aus Gründen der Übersichtlichkeit am besten wie im Programm an den Anfang jedes neuen Falls setzen.

Der stets als letzter stehende *ELSE*-Teil entspricht exakt seinem Zwilling aus der IF-Abfrage, er wird ausgeführt, falls die Variable (hier: *Kind*) einen zulässigen, aber im *CASE*-Teil nicht aufgeführten Wert annimmt. Ebenso wie bei der IF-Abfrage darf bei der *CASE*-Abfrage der *ELSE*-Teil fehlen. Während bei der IF-Abfrage das System aber bei fehlendem *ELSE*-Teil einfach zur ersten nachfolgenden Anweisung weitergeht, gibt es bei der *CASE*-Abfrage einen "Programmabsturz", falls der *ELSE*-Teil fehlt und ein "sonstiger" Fall trotzdem eintritt.

Werten Sie dies Programm bitte wie die vorigen mit Papier und Graphitstift aus, bevor Sie es ausführen lassen; Sie sollten sich, falls Ihr Ergebnis nicht mit der Bildschirmausgabe übereinstimmt, die Konstruktion der *CASE*-Abfrage noch einmal genau ansehen und auf die (ach so tückischen) Einzelheiten achten.

#### 4.5. Eingabe von Daten

Mit dem vorletzten Programm dieses Kapitels - Konvert.odc - gibt es etwas gegenüber den bisherigen Programmen Neuartiges, nämlich ein Programm, mit dem man als Benutzer kommunizieren kann, wenn

auch auf einem noch sehr einfachen Niveau. Sie finden in diesem Programm einerseits eine Wiederholung und Zusammenfassung dessen, was Sie in diesem Kapitel gelernt haben, Sie finden aber auch einige wichtige Ergänzungen Ihrer bisherigen Kenntnisse.

Sehen Sie die Importanweisung an. Sie finden außer dem bekannten Modul *Out* für die Datenausgabe das Modul *In*, mit dem sich, wie der Name vermuten läßt, Daten zur Verarbeitung einlesen lassen (markieren Sie in BlackBox den Modulnamen *In* und lassen Sie sich die Schnittstelle bzw. die Dokumentation des Moduls anzeigen). Weiter finden Sie diesmal nicht die übliche Kommandoprozedur *Start*, dafür die beiden Kommandos *Umrechnen* und *Anleitung*.

Beide Kommandos werden prinzipiell genauso ausgeführt, wie die bisherigen *Start*-Kommandos, Sie schreiben den Kommandonamen (z. B. *TutKonvert.Anleitung*) in das Log-Fenster oder an eine andere Stelle innerhalb des BlackBox Systems und lassen das Kommando ausführen. Die *Anleitung* teilt dem Benutzer den Zweck und die "Bedienungsweise" des Kommandos *Umrechnen* mit. Bevor Sie *Umrechnen* ausführen lassen, sollten Sie jedoch die Prozedur durcharbeiten, um das Component Pascal Verfahren zum Einlesen von Daten zu verstehen.

Im Deklarationsteil der Prozedur finden Sie eine Reihe Konstanten des Typs *REAL*, die Umrechnungsfaktoren für die anschließenden Berechnungen. Die ersten drei Variablendeklarationen sind Ihnen bekannt, es werden zwei Variablen des Typs *REAL* und eine Variable des Typs *CHAR* deklariert. Die anschließende Deklaration der Variablen *Sorte* stellt eine Neuigkeit dar, ihr Typ, *ARRAY 10 OF CHAR*, definiert eine Zeichenkette und ist kein einfacher, sondern ein sogenannter zusammengesetzter Typ. Damit ist eine Typdefinition gemeint, bei der ein neuer Typ als Kombination bekannter Typen vereinbart wird (auf die Existenz des Typs *ARRAY OF CHAR* als Typ für Zeichenketten (strings) wurde bereits kurz bei dem Programm ConstVar.odc hingewiesen, im Programm 7.5, String.odc, wird dieser Typ genau erläutert werden).

Wichtig und neu sind die im Anweisungsteil zu findenden Zeilen *In.Open* und *In.Char*. Mit *In.Char* wird ein einzelnes Zeichen eingelesen. Dies ist aber erst möglich, nachdem mit der Anweisung *In.Open* der sogenannte Einlesestrom geöffnet worden ist. Wenn Sie mit dem Modul *In* Daten für ein Programm einlesen wollen, müssen Sie in jedem Fall als erstes eine *In.Open*-Anweisung ausführen lassen.

Die Schleife *REPEAT In.Char(Hilf) UNTIL (Hilf > 020X) OR ~In.Done* liest solange ein (einzelnes) Zeichen nach dem anderen vom Einlesestrom in die Variable *Hilf*, bis entweder ein Zeichen größer als *020X* gelesen wurde, was im Klartext bedeutet, das der Einlesevorgang nach dem Erreichen desjenigen Zeichens beendet wird, das in der Latin1-Tabelle eine höhere Platznummer als das Leerzeichen hat, oder aber das Ende des Einlesestroms erreicht wurde, in diesem Fall erhält die Variable *In.Done* den Wert *FALSE* zugewiesen.

Die vom Modul *In* exportierte Boole'sche Variable *In.Done* bietet ein bemerkenswert einfaches Verfahren, die Korrektheit eines Einlesevorgangs zu prüfen. Diese Variable wird vom Modul *In* beim Öffnen

des Einlesestroms mit dem Kommando *In.Open* auf den Wert *TRUE* gesetzt. Nach jedem Lesevorgang wird der Wert neu festgesetzt. Er bleibt also, vereinfacht gesagt, solange *TRUE*, wie das Einlesen der Daten fehlerfrei ist und wird *FALSE*, wenn ein Lesefehler auftritt, was beispielsweise geschieht, wenn das Ende des Einlesestroms erreicht ist, also kein weiteres Zeichen gelesen werden kann.

Die bei Ihnen inzwischen sicher aufgetauchte Frage, wo denn der Einlesestrom zu finden sei, soll jetzt beantwortet werden. BlackBox bietet in dieser Hinsicht eine große Freizügigkeit, Sie können die einzulesenden Daten an irgendeine Stelle innerhalb der Systemumgebung schreiben, ebenso wie Sie das mit den Kommandonamen bei der Ausführung von Kommandos getan haben. Der Anfang des Einlesestroms ist für BlackBox beim Öffnen des Einlesestroms mit der Anweisung *In.Open* immer durch das erste (von Ihnen) markierte Zeichen (bei fehlender Markierung durch das erste Zeichen im aktiven Fenster) gegeben. Sie können also die für die Prozedur *Umrechnen* einzulesenden Daten entweder vollständig oder teilweise markieren (das erste Zeichen genügt), oder auch nur irgendeins der davor stehenden Leerzeichen, aber auch (durch Doppelklick am Zeilenanfang) die gesamte die Daten enthaltende Zeile, den Rest erledigt BlackBox. Sofern Sie die am Ende des Programms stehende Zeile mit der Zeichenfolge *Z 4.3* (oder einen geeigneten Teil von ihr) markiert haben, enthält die Variable *Hilf* nach dem Ende der *REPEAT*-Schleife den Buchstaben *Z*.

Die der *REPEAT*-Schleife folgende Mehrfachverzweigung besteht aus einem *CASE*- und einem *ELSE*-Teil. Der *ELSE*-Teil ist aus folgenden Gründen nötig. Es kann sein, daß entweder der Benutzer vergessen hat, die einzulesenden Daten zu markieren, oder das (erste) eingelesene Datum ist ein "falsches" Zeichen. Als "falsch" werden vom *CASE*-Teil alle Zeichen außer "E", "F", "M" oder "Z" bewertet, nur diese Buchstaben führen zur Verarbeitung der Anweisungen dieses Teils. Der *ELSE*-Teil wird für den Fall benötigt, daß ein anderes oder kein Zeichen gelesen wurde. Es gäbe sonst, wie Sie wissen, einen vom System veranlassten Programmabbruch.

Im *CASE*-Teil wird als erstes durch die Anweisung *In.Real(Betrag)* der umzurechnende Zahlenwert in die Variable *Betrag* gelesen. Ähnlich wie das Modul *Out* für jeden Variablentyp eine eigene Ausgabe-prozedur enthält, gibt es im Modul *In* für jeden Variablentyp gesonderte Einleseprozeduren. Der *IF In.Done THEN*-Teil der Verzweigung wird daher nur durchlaufen, wenn tatsächlich als zweiter Parameter des Einlesestroms eine (*REAL*- oder *INTEGER*-) Zahl gelesen wurde, sonst wird eine Fehlermeldung ausgegeben, denn *In.Done* enthält in diesem Fall den Wert *FALSE*.

Die Ausgabeanweisungen der zweiten (inneren) *CASE*-Sequenz müssen nicht weiter analysiert werden, sie geben in Abhängigkeit vom ersten gelesenen Parameterwert, dem Anfangsbuchstaben der umzurechnenden Längeneinheit, das dem zweiten gelesenen Parameterwert, der Maßzahl, entsprechende Ergebnis in der international standardisierten Einheit Meter aus.

Verdeutlichen Sie sich bitte, daß die äußere *CASE*-Abfrage zusammen mit der *IF In.Done*-Bedingung alle möglichen Eingabefehler abfängt, die innere *CASE*-Abfrage daher keinen *ELSE*-Teil benötigt. Sie se-

hen, wie wichtig es ist, mögliche "falsche" Handlungen eines Programmbenutzers durch geeignete Konstruktionen unschädlich zu machen, so wie im Falle der Maßeinheit die Eingabe eines nicht vorgesehenen Anfangsbuchstabens durch den *ELSE*-Teil der *CASE*-Abfrage wirkungslos gemacht worden ist, ebenso die mögliche Eingabe eines ungeeigneten Zahlwerts durch das Zusammenwirken der *IF*-Abfrage und der Variablen *In.Done*. Bei Ihrer eigenen Programmierarbeit sollten Sie derartige potentielle Fehleingaben immer berücksichtigen und entsprechende Abfangmechanismen einbauen. Versetzen Sie sich bei der Arbeit an einer Programmidee am besten in die Situation, in der Sie sich möglicherweise oft genug befunden haben, in die Situation des Benutzers. Wenn Sie zusätzlich annehmen, daß dieser Benutzer (anders als Sie und ich) Bedienungsanleitungen prinzipiell nicht liest, im Umgang mit dem Programm nachlässig und unkonzentriert ist, dafür aber äußerst experimentierfreudig, werden Sie mit dem fertigen Produkt Ihrer Bemühungen vermutlich den wenigsten Ärger haben.

Mit *Konvert.odc* haben Sie eine Möglichkeit kennengelernt, ein Programm zu erstellen, das (nachträglich) eingegebene Daten verarbeiten kann. Die Form der Dateneingabe ist durch die Verwendung des absichtsvoll einfachen Moduls *In* für das Erlernen einer Programmiersprache wie *Component Pascal* gut geeignet, da der Einlesemechanismus sehr durchsichtig ist, sie entspricht aber sicher nicht den bei einem vollwertigen Programm erwarteten Verfahren.

Als professionelles Programmentwicklungssystem bietet BlackBox selbstverständlich Möglichkeiten, die für solche Zwecke üblichen Eingabemasken zu erstellen. Solche Masken oder Fenster werden in der Informatik auch als "graphische Benutzerschnittstellen" (*Graphical User Interfaces*, *GUI*) bezeichnet. Als Standard-Datenmasken existieren in den meisten kommerziellen Programmen sogenannte *Dialog-Boxen* oder *Dialoge*. Die nachstehende Abbildung zeigt als Beispiel die Dialogbox, die Sie erstellt haben werden, wenn Sie am Ende des Programms *Konvert1.odc* angelangt sind.



Abbildung 1  
Dialogbox zum Modul *TutKonvert1*

Bei einem flüchtigen Blick auf den Quelltext des Programms `Konvert1.ode` kann es Ihnen geschehen, daß Sie keinen Unterschied zu dem vorigen Programm `Konvert.ode` entdecken. Der erste, bei einem genaueren Blick auftauchende Unterschied findet sich in der Importanweisung, in der das eben noch für den Einlesevorgang wesentliche Modul `In` fehlt. Wie sollen also die Daten in das Programm transportiert werden? An dieser Stelle kommen die vom BlackBox System zur Verfügung gestellten Dienstmodule und Metaprogrammierungsmöglichkeiten ins Spiel, auf deren Wirkungsweise im Folgenden eingegangen wird.

Sie sehen in der Dialogbox der Abbildung 1 vier Schaltflächen mit den Bezeichnern `Zoll`, `Fuß`, `Ellen`, `Meilen`. Diese Bezeichner finden Sie im Modul `TutKonvert1` zweimal wieder, als globale Konstanten im Deklarationsteil des Moduls und - fast genauso geschrieben - als lokale Konstanten im Deklarationsteil der Prozedur `Umrechnen`. Beachten Sie bitte, daß diese Bezeichner im einen Fall kleine, im anderen Fall große Anfangsbuchstaben haben. Innerhalb desselben Moduls (genau genommen innerhalb desselben Sichtbarkeitsbereiches, den Unterschied lernen Sie im folgenden Kapitel kennen) darf jeder Bezeichner nur einmal vorkommen. Da der unterschiedlichen Schreibweise der Anfangsbuchstaben unterschiedliche Zahlencodes der zugeordneten Zeichen entsprechen, werden in der sonstigen Schreibweise identische Bezeichner in Component Pascal als voneinander verschieden gewertet.

Außer den Konstanten sehen Sie im Deklarationsteil von `TutKonvert1` die drei Variablen `Betrag`, `sorte` und `Sorte` deklariert, von denen `Sorte` nicht mit einem "\*", sondern mit einem "-" als Exportmarke versehen ist; der Unterschied zwischen beiden Exportmöglichkeiten wird auf den folgenden Seiten erklärt werden. Die Variablenamen sind Ihnen teilweise aus dem vorigen Programm bekannt, die letzten beiden verdeutlichen noch einmal die Tatsache, daß Bezeichner mit unterschiedlicher Schreibweise der (Anfangs-) Buchstaben in Component Pascal nicht identisch sind.

Eine Neuigkeit ist das Erscheinen eines Deklarationsteils im Modulrumpf. Prinzipiell ist ein Modul genauso strukturiert wie eine Prozedur. Das minimale Modul `TutA` (Programm 1.1) war folglich zwar die Wahrheit, aber eben nicht die ganze Wahrheit, ein Modul kann außer Prozeduren weitere Deklarationen enthalten und einen Anweisungsteil besitzen, beides sehen Sie im Modul `TutKonvert1`. Neu ist außerdem, daß nicht nur Prozedurnamen, sondern auch andere im Modul deklarierte Bezeichner exportiert werden. Grundsätzlich gilt allerdings in Component Pascal, daß nur Bezeichner exportiert werden können, die "global" sind, was nicht anderes heißt, als daß sie im Modulrumpf deklariert sein müssen. Ein Versuch, "lokale" Bezeichner, solche Bezeichner also, die in einer Prozedur deklariert sind, zu exportieren, würde vom Compiler abgewiesen werden (Näheres zu "globalen" und "lokalen" Bezeichnern erfahren Sie im nächsten Kapitel).

Der Anweisungsteil eines Moduls wird ebenso wie der Anweisungsteil einer Prozedur mit dem Schlüsselwort `BEGIN` eingeleitet und steht hinter allen Deklarationen, wobei zu den Deklarationen auch die Prozeduren gehören. Der Anweisungsteil des Moduls `TutKonvert1` wird also von der am Ende des Programms unterhalb der Kommentarzeile `Initialisierung` stehenden Anweisung gebildet. Der Anweisungsteil

eines Moduls hat eine spezielle Eigenschaft, er wird vor allen Prozeduren abgearbeitet, unmittelbar nachdem das Modul in den Arbeitsspeicher des Rechners geladen worden ist. Da ein Modul solange geladen bleibt, bis es explizit entladen wird, wird der Anweisungsteil des Moduls ein einziges Mal beim Laden des Moduls ausgeführt. Wie durch den Kommentar angedeutet, dient der Anweisungsteil eines Moduls deshalb in erster Linie dazu, Variablen des Programms zu initialisieren.

Außer dem von Prozeduren bekannten `BEGIN`-Teil kann der Anweisungsteil eines Component Pascal Moduls einen weiteren (in `TutKonvert1` nicht vorhandenen) Anweisungsblock enthalten, der mit dem Schlüsselwort `CLOSE` beginnt. Dieser Anweisungsblock wird beim expliziten Entladen eines Moduls nach der Beendigung aller Prozeduren abgearbeitet und spielt in großen Programmen mit vielen voneinander abhängigen Modulen eine Rolle.

Mit diesen Erläuterungen dürfte das Modul `TutKonvert1` wegen seiner Ähnlichkeit zu `TutKonvert` verständlich sein, unklar bleibt aber die Frage, wie die Daten in das Programm kommen - zu diesem Zweck dienen die bereits erwähnten Dialogboxen. Zur Erstellung einer Dialogbox für das Modul `TutKonvert1` klicken Sie im Menü `Controls` auf `New Form...`, es erscheint die folgende Dialogbox

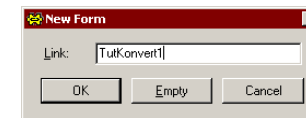


Abbildung 2  
Erzeugung eines Neuen Dialogs für  
`TutKonvert1`

Schreiben Sie in das Feld `Link` den Modulnamen `TutKonvert1` und klicken Sie anschließend auf `OK` (falls Sie mit einem MacIntosh-Rechner arbeiten, heißt die Schaltfläche `Create`). Die Dialogbox verschwindet und Sie erhalten statt dessen den Dialog

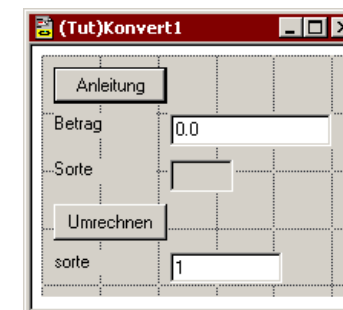


Abbildung 3  
Von BlackBox generierter Dialog für  
`TutKonvert1`

Wie Sie an den Titelleisten der Boxen sehen, sind diese übliche Windows-Fenster (auf einem Macintosh-Rechner sehen die Boxen etwas anders aus als hier abgebildet, nämlich wie übliche MacOS-Fenster).

Wenn Sie die Beschriftungen, Schaltflächen und Fenster der Abbildung 3 mit den vom Modul *Tut-Konvert1* exportierten Bezeichnern vergleichen, werden Sie feststellen, daß diese identisch sind. BlackBox liest die Exporte aus dem kompilierten Modul und erstellt damit ein Standard-Layout für die Dialogbox. In dieser sehen Sie zwei Schaltflächen mit den Namen der beiden Kommandoprozeduren des Moduls, außerdem drei Felder, vor denen als Bezeichner die Namen der drei exportierten Variablen stehen.

Zwei der Felder enthalten bereits Einträge, das dritte, der Variablen *Sorte* zugeordnete Feld ist leer und andersfarbig unterlegt. Die andere Hintergrundfärbung signalisiert, daß Sie in dieses Feld keine Daten eingeben können und hängt mit einer speziellen Component Pascal Eigenschaft zusammen, die schreibgeschützter Export (read only export) genannt wird. Wie bereits erwähnt, wird die Variable *Sorte* nicht mit einem "\*" exportiert, sondern mit einem "-", dies ist das reservierte Zeichen für schreibgeschützten Export. Die Nützlichkeit dieser Exportform werden Sie erst in späteren Programmen erkennen können, es handelt sich um ein sehr effizientes Mittel, potentiell gefährdete Daten eines Moduls vor äußerer Veränderung zu bewahren.

Folge des schreibgeschützten Exports ist, daß Sie im Gegensatz zu den beiden anderen Datenfeldern in dem Feld *Sorte* keine Eintragungen machen können, es dient ausschließlich zur Anzeige der vom Programm bei der Ausführung des Kommandos *Umrechnen* ermittelten Längeneinheit. Die anderen beiden Felder sind dagegen nicht schreibgeschützt, es können - und müssen - in ihnen Eintragungen vorgenommen werden, wie Ihnen die Prozedur *Anleitung* mitteilt.

Klicken Sie auf die Schaltfläche *Anleitung* erscheint allerdings nicht die erwartete Ausgabe der Bedienungsanleitung im Log-Fenster, vielmehr wird die Schaltfläche umrahmt und mit "Anfassern" versehen, sie ist markiert. Dialoge werden in BlackBox bei der Erstellung im sogenannten Bearbeitungsmodus (*Layout Mode*) geöffnet. Sie können in diesem Modus die einzelnen Bestandteile des Dialogs markieren, verschieben, in der Größe oder der Beschriftung verändern, die Gestaltung der Dialogbox ist Ihnen völlig freigestellt.

Wollen Sie den Dialog zur Bedienung des Programms benutzen, müssen Sie zuerst in den Maskenmodus (*Mask Mode*) umschalten. Klicken Sie dazu im Menü *Dev* auf *Mask Mode*. Das Aussehen der Dialogbox ändert sich, das Hintergrundgitter, das bisher den Bearbeitungsmodus angezeigt hat, ist verschwunden, die Box befindet sich im Maskenmodus. Klicken Sie jetzt erneut auf die Schaltfläche *Anleitung*, erscheint wie erwartet im Log-Fenster der von der gleichnamigen Prozedur ausgegebene Text.

Ebenso können Sie jetzt auf die Schaltfläche *Umrechnen* klicken, im Log-Fenster erscheint der Text *0.0 englische Zoll entsprechen 0.0 Meter(n)*. BlackBox hat die Berechnung mit den Daten der Variablen *Betrag*, *sorte* und *Sorte* vorgenommen. Das Feld *Sorte* enthält allerdings nicht die Angabe *Zoll*, sondern ist weiterhin leer, obwohl von der CASE-Abfrage der Prozedur *Umrechnen* der Wert der Variablen *Sorte* ent-

sprechend dem Wert von *sorte* eingesetzt worden ist und *sorte* beim Laden des Moduls mit *zoll (=1)* initialisiert wurde. Dies liegt daran, daß die Bildschirmdarstellung nicht automatisch aufgefrischt worden ist. Sie können eine Aktualisierung am einfachsten erreichen, indem Sie das Dialogfenster minimieren und wieder öffnen (im 9. Kapitel erfahren Sie, wie Sie im BlackBox System die automatische Aktualisierung der Daten eines Dialogs erreichen können).

Der Eintrag 0.0 im Feld *Betrag* beruht darauf, daß bei allen global deklarierten Variablen vom System für jeden Typ eine Standardinitialisierung erfolgt. Bei Zahltypen ist das der Wert Null, wie am Feld *Betrag* ersichtlich. Dieser wurde in der Berechnung verwendet, da bei der Initialisierung des Moduls keine Zuweisung eines Wertes erfolgt ist und Sie diesen bisher auch nicht durch Eingabe einer anderen Zahl in das Feld *Betrag* geändert haben. Bei Zeichenkettentypen ist die Initialisierung, wie Sie an der Variablen *Sorte* gesehen haben, durch eine leere Zeichenkette der Länge Null Zeichen gegeben.

Wenn Sie jetzt in das Feld *sorte* die Zahl "4" als Codenummer für die Längeneinheit "Meilen" und in das Feld *Betrag* den Wert "3.5" als umzurechnende Länge eingeben, erscheinen nach einem Klick auf die Schaltfläche *Umrechnen* im Log-Fenster der Text *3.5 englische Meile(n) entsprechen 5632.704 Meter(n)* und im Feld *Sorte* (nach Minimieren und Wiederöffnen der Dialogbox) die Zeichenfolge "Meile(n)".

Wie Sie sehen, hat das BlackBox System die Fähigkeit, mit Hilfe von Dialogen Daten, wie hier die Längeneinheit und den Betrag der umzurechnenden Länge, einzulesen und auszugeben, ohne daß Sie als Programmierer eine explizite Angabe der dazu benötigten Module, etwa in Form einer Importanweisung, machen müssen. Sie brauchen also für die Erstellung einfacher Datenmasken keine Kenntnis komplizierter Modulbibliotheken, BlackBox ist ein sogenanntes RAD (Rapid Application Development) System, dessen ganze Kapazitäten Sie zwar nur nutzen können, wenn Sie in der Lage sind, die Schnittstellen der wichtigen Dienstmodule zu benutzen, das Ihnen aber bei der Erstellung "normaler" Anwendungen die meiste Arbeit abnimmt.

Geben Sie in das Feld *sorte* eine andere Zahl als "1", "2", "3" oder "4" ein, erhalten Sie zwar einerseits im Log-Fenster die im *ELSE*-Teil der CASE-Abfrage stehende Fehlermeldung, andererseits erhalten Sie allerdings unabhängig davon eine "Umrechnung" mit unvorhersehbaren Ergebnissen, da die Prozedur *Umrechnen* keinen Schutz gegen die Eingabe einer anderen Zahl enthält. Statt jedoch das Modul selbst zu ändern, können Sie den Schutz gegen "falsche" Werte von *sorte* auch durch die nachstehend beschriebene Änderung der Datenmaske erreichen, die durch ihren Aufbau die möglichen Werte der Variablen *sorte* auf das vorgesehene Intervall "1" bis "4" begrenzt.

Im folgenden sollen Sie die Dialogbox der Abbildung 1 als Beispiel für einige der in BlackBox vorhandenen Gestaltungsmöglichkeiten selbst erstellen. Klicken Sie dazu erneut im Menü *Controls* auf *New Form...*, anschließend geben Sie jedoch keinen Modulnamen an, sondern klicken auf *Empty*. Fügen Sie in das sich öffnende leere Fenster der Reihe nach aus dem Menü *Controls* mit den angegebenen Kommandos die folgenden Kontrollelemente ein. Sie benötigen vier Auswahl Schaltflächen (*Insert Radio Button*), ein

Eingabefeld (*Insert Edit Field*) und zwei Kommandoschaltflächen (*Insert Command Button*), außerdem zwei Beschriftungsfelder (*Insert Caption*). Alle Kontrollelemente tragen, sofern sie beschriftet sind, entweder die Beschriftung *untitled* oder *Caption*. Arrangieren Sie die Kontrollelemente nach eigenem Geschmack oder entsprechend der Anordnung der Abbildung 1. Es wird Ihnen auffallen, daß alle Kontrollelemente bis auf die mit *Caption* beschrifteten in gleicher Weise wie im vorigen Dialog das Feld *Sorte* als passiv gekennzeichnet sind. Es fehlen ihnen die Zuordnungen entsprechender Aktivatoren, die Sie als Nächstes vornehmen müssen. Markieren Sie eine der Schaltflächen mit der Beschriftung *untitled* und klicken Sie im Menü *Edit* auf *Object Properties...* (MacIntosh: *Part Info...*) oder wählen Sie im Kontextmenü, zweite [rechte] Maustaste den Eintrag *Properties*. Es öffnet sich der sogenannte Objektinspektor (s. Abb. 4), mit dem Sie die Eigenschaften eines Kontrollobjekts festlegen können. Das erste, mit *Control* beschriftete Feld ist schreibgeschützt und enthält als Eintrag die Bezeichnung des inspezierten Kontrollelements, in diesem Fall also *Command Button*.

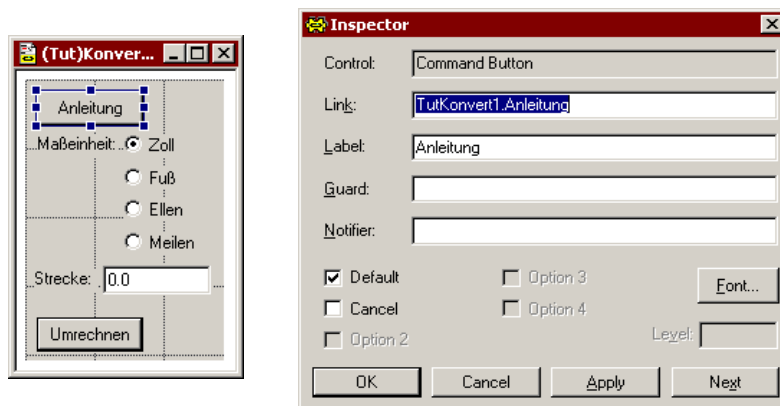


Abbildung 4  
Inspektor-Dialog für die Attribute einer Dialogbox

Es sollen jetzt nicht alle Felder des Inspektors erklärt werden, im Augenblick benötigen Sie nur die beiden nächsten, *Link* und *Label*. Das Feld *Link* ist das wichtigste Feld des Inspektors, mit ihm teilen Sie dem System mit, wo die zu verarbeitenden Größen zu finden sind. In dieses Feld tragen Sie den qualifizierten Bezeichner des zugehörigen Kommandos ein, *TutKonvert1.Anleitung*. In das Feld *Label* tragen Sie die gewünschte Bezeichnung *Anleitung* ein; klicken Sie anschließend auf *Apply* (MacIntosh: *Set*). Markieren Sie bei geöffnetem Inspektor als nächstes die zweite Schaltfläche. Ersetzen Sie die Beschriftung *untitled* des Feldes *Label* durch *Umrechnen*, tragen Sie in das Feld *Link* das Kommando *TutKonvert1.Umrechnen*

ein und klicken Sie wieder auf *Apply*. Damit sind beide Schaltflächen mit den entsprechenden Kommandos des Moduls *TutKonvert1* verbunden und könnten bereits aktiviert werden.

Allerdings gibt es bisher für das Kommando *Umrechnen* keine Parameter, da keine Felder für die Daten festgelegt wurden. Markieren Sie das von Ihnen in die Dialogbox eingefügte Eingabefeld, im Inspektor tragen Sie in das Feld *Link* die Variable *TutKonvert1.Betrag* ein, das Feld *Label* können Sie frei lassen; klicken Sie auf *Apply*. Die Darstellung des Feldes ändert sich als Zeichen dafür, daß es mit der nicht schreibgeschützten Variablen *TutKonvert1.Betrag* verbunden und in den aktivierten Zustand übergegangen ist, Sie können ab jetzt Einträge in das Feld vornehmen.

Sofern Sie sich an das Layout der Abbildung 1 gehalten haben, steht vor dem eben bearbeiteten Eingabefeld eines der beiden Schriftfelder mit der vorläufigen Bezeichnung *Caption*. Klicken Sie als nächstes auf dieses Feld und ändern Sie im Inspektor den Eintrag im Feld *Label* in *Strecke*., in das Feld *Link* können Sie außerdem als Erinnerungshilfe den Namen des zugehörigen Feldes *TutKonvert1.Betrag* eingeben, der Eintrag ist für das System aber nicht erforderlich; klicken Sie auf *Apply*. Ändern Sie analog den Bezeichner des zweiten, oberen Schriftfeldes in *Maßeinheit*: und klicken Sie wieder auf *Apply*.

Bei der anschließend nötigen Bearbeitung der vier Auswahlschaltfelder gibt es eine Neuigkeit. Sie müssen allen vier die Variable *TutKonvert1.sorte* zuordnen, denn jedes der vier Felder soll die Wahl einer der Maßeinheiten ermöglichen, die die Variable *sorte* im Programm annehmen kann. Um die Maßeinheiten unterscheiden zu können, muß jedes der Felder ein weiteres, eindeutiges Merkmal erhalten. Klicken Sie auf das oberste Auswahlschaltfeld, im *Link*-Feld des *Inspektor*-Dialogs tragen Sie die zugeordnete Variable *TutKonvert1.sorte* ein und in das *Label*-Feld den Bezeichner *Zoll*. In das unten rechts im Inspektor stehende Feld *Level*: tragen Sie eine "1" ein, diejenige Ganzzahlkonstante, die im Modul *TutKonvert1* dem Bezeichner *zoll* zugeordnet ist. Entsprechend ordnen Sie den übrigen Auswahlschaltfeldern die passenden Zahl-Bezeichner-Kombinationen zu (vergessen Sie nicht, nach jeder Zuordnung auf *Apply* zu klicken, ihre Einträge werden sonst nicht wirksam). Alle vier Auswahlschaltfelder sind auf diese Weise einerseits der Variablen *TutKonvert1.sorte* zugeordnet worden, andererseits wirken sie praktisch wie Konstanten, da jedes Feld, das der Benutzer anklickt, *TutKonvert1.sorte* denjenigen Wert zuweist, der über das Inspektorfeld *Level* der Maßeinheit zugeordnet wurde.

Schließen Sie zum Testen Ihrer Arbeit den Inspektor. Um die Größe des Dialogs der Anordnung der Kontrollelemente anzupassen, wählen Sie den Menüpunkt *Layout* → *Recalc Focus Size*; alternativ können Sie *Edit* → *Select Document* wählen und die Größe des Dialogs manuell festlegen.

Klicken Sie jetzt auf *Controls* → *Open as Aux Dialog*, öffnet sich eine Kopie des eben erstellten Dialogs im Maskenmodus, mit der Sie dessen Funktionen erproben können. Sollte eine dieser Funktionen nicht wie gewünscht arbeiten, können Sie die entsprechenden Änderungen in demjenigen Dialog vornehmen, der sich im Bearbeitungsmodus befindet und die Änderung simultan im anderen Dialog erproben. Sie müssen Ihre Arbeit nicht zwischenspeichern, jede Änderung, nicht nur eine Layout-Änderung, sondern

beispielsweise auch die Änderung der umzurechnenden Längeneinheit im maskierten Fenster wird gleichermaßen in beiden Dialogen angezeigt.

Wenn Sie mit dem Ergebnis Ihrer Arbeit zufrieden sind, können Sie die Dialogbox speichern. Standardmäßig zählt ein solcher Dialog zu den Ressourcen eines (Unter-) Systems und sollte deshalb im entsprechenden Verzeichnis gespeichert werden, in diesem Fall also im Unterverzeichnis *Rsrc* des Verzeichnisses *Tut*. Wollen Sie die Dialogbox benutzen, können Sie sie jederzeit im Maskenmodus öffnen (weitere Möglichkeiten von Dialogen erklärt Ihnen das neunte Kapitel dieses Tutoriums. Ausführliche Informationen über die Dialogerstellung finden Sie im Kapitel 4 - *Forms* - und im Kapitel 6 - *View Construction* - von *A Case Study using BlackBox Components*, der Dokumentation zum BlackBox System).

Ein wesentlicher Aspekt von Programmen, der bisher nur gestreift wurde, ist die Programmdokumentation. Dabei sind zwei Formen zu unterscheiden, die Dokumentation der Programmentwicklung und die Dokumentation der Funktionalität eines Programms. Den Benutzer eines Programms wird die Entwicklungsarbeit nicht interessieren, allerdings erwartet er vom Entwickler und Verkäufer, daß dieser (immer vorhandene) Fehler des Programms beseitigt und es neu auftauchenden Anforderungen anpaßt, aus diesem Grund benötigt jeder Entwickler außer den Quelltexten der Programme eine eigene ausführliche und gegliederte Dokumentation der Geschichte des Programms, wobei einer der wesentlichsten Punkte die Darstellung der bei der Entwicklung aufgetauchten Programmfehler ist.

Für den Benutzer eines Programms wichtig ist selbstverständlich eine gute Dokumentation der Fähigkeiten, die das Programm hat, wobei eine klare Darstellung der Voraussetzungen dazu gehört, ohne die das Programm nicht oder nur eingeschränkt die erwarteten Leistungen erbringt. BlackBox enthält als Entwicklungsumgebung für Anwenderprogramme mit der oben erwähnten Dokumentation *A Case Study using BlackBox Components* eine gute Darstellung der Grundlagen und Anwendungsmöglichkeiten des Systems. Darüber hinaus bietet der Ihnen inzwischen bekannte *Interface Browser* (Menü *Info* → *Client Interface*) eine bemerkenswert einfache Möglichkeit, sich schnell und unkompliziert mit der Schnittstellenbeschreibung eines einzelnen Moduls eine Übersicht seiner Exporte und damit seiner Fähigkeiten zu verschaffen.

*DEFINITION TutKonvert1;*

*VAR*

*Betrag: REAL;*  
*Sorte: ARRAY 10 OF CHAR;*  
*sorte: INTEGER;*

*PROCEDURE Anleitung;*  
*PROCEDURE Umrechnen;*

*END TutKonvert1.*

**Schnittstelle des Moduls *TutKonvert1***

Die Schnittstellenbeschreibung des Moduls *TutKonvert1* sagt Ihnen, welche Variablen und Prozeduren das Modul exportiert, es ist jedoch klar, daß sich mit diesen Informationen allein eine Dialogbox wie die von Ihnen erstellte nicht oder nur nach vielen Irrtümern und Experimenten konstruieren läßt. Über die von BlackBox gelieferte Schnittstelle hinaus muß es eine Dokumentation geben, in der die Bedeutung und Funktionsweise der Exporte dargestellt wird. BlackBox enthält mit *Documentation Conventions* (Menü *Help* → *Contents* → *Documentation Conventions*) eine Orientierungshilfe für die Erstellung von Dokumentationen, die Sie Ihrer eigenen Arbeit zugrunde legen sollten. Am Beispiel des Moduls *TutKonvert1* möchte ich im folgenden darstellen, wie eine Dokumentation, die sich an den BlackBox Konventionen orientiert, aussehen kann.

Öffnen Sie dazu die Dokumentation des Moduls *TutKonvert1* indem Sie den Modulnamen in der BlackBox Umgebung markieren und den Menüpunkt *Info* → *Documentation* wählen. Standardisiert werden die Dokumentationen eines BlackBox (Unter-) Systems im Unterverzeichnis *Docu* des jeweiligen (Unter-) Systems gespeichert. Entsprechend befindet sich die Dokumentation des Moduls *TutKonvert1* als Datei *Konvert1.odc* im Unterverzeichnis *Tut/Docu* (nicht zu verwechseln mit der gleichnamigen Quelltextdatei im Unterverzeichnis *Tut/Mod*). Die Dokumentation wird eingeleitet mit der vom Browser erzeugten Schnittstellenbeschreibung gefolgt von einer Darstellung der wesentlichen Arbeits- und Funktionsweise des Moduls. Es folgen die Erläuterungen aller Bezeichner der Schnittstellenspezifikation in der Reihenfolge ihrer Auflistung.

Zwei Besonderheiten seien hier herausgehoben. Die Erklärung der Variablen *sorte* enthält als Zusatz zur Auflistung der Variablendeklaration die für ihre korrekte Benutzung erforderliche Einschränkung des Variablentyps auf das Intervall  $1 \leq \text{sorte} \leq 4$  in semiformaler Schreibweise. Einen ähnlichen Zusatz, der etwas ausführlicher erläutert werden soll, finden Sie im Anschluss an die Erklärung der Prozedur *Umrechnen*. Wie bereits angedeutet, erwartet jedes Programm die Erfüllung gewisser Voraussetzungen für sein Funktionieren, wofür es (idealerweise) bestimmte Leistungen garantiert. Dieses Verständnis von Programmen wird *programming by contract* genannt, da es den Bedingungen eines Vertrages zwischen zwei Geschäftsparteien ähnelt. Der Auftraggeber sichert die Existenz gewisser Voraussetzungen zu, der Auftragnehmer verpflichtet sich, unter diesen Voraussetzungen eine bestimmte Leistung zu erbringen, Voraussetzung und Verpflichtung werden in einem verbindlichen Kontrakt festgehalten. Die Voraussetzungen werden auch Präkonditionen (preconditions) genannt, die zu erbringenden Leistungen bezeichnet man entsprechend als Postkonditionen (postconditions).

Diese Vertragssituation gilt nicht nur für Programme, sondern gleichermaßen für Prozeduren, jede Prozedur benötigt für ein korrektes Funktionieren die Erfüllung gewisser Voraussetzungen und sie liefert nach ihrem Ablauf ein bestimmtes Ergebnis, sofern die Eingangsvoraussetzungen erfüllt waren. Derartige Bedingungen und Ergebnisse werden in der semiformalen Schreibweise der BlackBox Dokumentationen mit *Pre* und *Post* gekennzeichnet. Entsprechend sehen Sie als Zusatz zu der Prozedur *Umrechnen* die von

dem Wort *Pre* eingeleitete Darstellung der Bedingungen, die für das einwandfreie Funktionieren der Prozedur erfüllt sein müssen und anschließend nach dem Wort *Post* die Zusicherungen, die bei Einhalten der Vorbedingungen von der Prozedur *Umrechnen* erfüllt werden (eine ausführliche Darstellung von Vor- und Nach-Bedingungen finden Sie in den erwähnten *Documentation Conventions*).

Das Prinzip der Moduldokumentationen läßt sich in einem Satz zusammenfassen. Programmieren Sie nicht in (realer oder vermeintlicher) Kenntnis einer Modulimplementierung, erstellen Sie Ihre Programme immer nur mit dem Blick auf die Schnittstellen der von Ihnen verwendeten Module und Programmieren Sie Ihre Module so, daß deren Schnittstellen alles Wesentliche dokumentieren. Im Englischen wird diese Regel *program to an interface* genannt, sie ist ein guter Leitfaden für das eigene Handeln. Gewöhnen Sie sich aber gleichzeitig an die Regel, zu einem Programm eine Dokumentation zu erstellen, denn keine noch so gut entworfene Schnittstelle kann eine Moduldokumentation ersetzen.