APPLICATION
SYSTEMS
HEIDELBERG

TUTORIAL UND ENTWICKLERHANDBUCH

REALbasic®

REALbasic Tutorial und Entwicklerhandbuch

Deutsche Übersetzung von O. Buchmann, T. Hoffmann, V. Ritzhaupt, M. Taubert, C. Schweinfurth

Postanschrift: Application Systems Heidelberg Software GmbH
Pleikartsförsterhof 4/1
D – 69124 Heidelberg, Deutschland

Telefon: 06221 300002

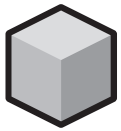
Telefax: 06221 300389

Web Site: http://www.application-systems.de/realbasic
http://www.realsoftware.com

Support: realbasic@application-systems.de

Verkauf: sales@application-systems.de

Stand dieser Ausgabe: September 2004



REALbasic®

TUTORIAL

Kapitel 0

Tutorial

1. Einführung in REALbasic

Willkommen bei REALbasic

REALbasic ist eine integrierte Entwicklungsumgebung, die auf einer modernen Form der BASIC-Programmiersprache basiert. Die REALbasic Entwicklungsumgebung beinhaltet eine objektorientierte Programmiersprache, einen Objekt-Browser, einen Quelltext-Debugger und eine umfangreiche Sammlung von Objekten zum Programmieren grafischer Benutzeroberflächen (**G**raphical **U**ser **I**nterface, GUI).

Damit stellt REALbasic alle Werkzeuge zur Verfügung, die Sie zum Entwickeln beliebiger Anwendungen benötigen.

Einsteiger werden viel Freude daran haben, Mac OS-, Windows- und Linux-Programme mit REALbasic zu entwickeln, die alle modernen Funktionen des Betriebssystems nutzen können.

Fortgeschrittene und erfahrene Programmierer werden vor allem den großen Umfang und die Qualität der in REALbasic integrierten Werkzeuge zu schätzen wissen.

Wie man dieses Tutorial benutzt

Dieses Tutorial bietet eine Reihe von praktischen Übungen, um REALbasic kennenzulernen. Die Lektionen sind so strukturiert, dass jede innerhalb einer halben Stunde absolviert werden kann. Da jedes Kapitel auf dem vorigen aufsetzt, sollten Sie das Tutorial in der richtigen Reihenfolge durcharbeiten.

Im Tutorial werden Sie mit REALbasic ein vollständiges Programm entwickeln. Sie werden einen Texteditor programmieren, der SimpleText, dem Texteditor, der mit Macintosh-Computern mitgeliefert wird, ähnlich ist. Mit REALbasic sind Sie in der Lage, Programme für das „klassische" Mac OS, Mac OS X, Windows und Linux zu compilieren.

Sie werden schnell die Fähigkeiten und die leichte Bedienbarkeit von REALbasic schätzen lernen. Für das gesamte Programm werden Sie nur ungefähr 200 Zeilen schreiben müssen, während SimpleText im Original aus etwa 20000 Zeilen C/C++ Code besteht.

Wer dieses Tutorial benutzen sollte

Dieses Tutorial wurde für Programmiereinsteiger geschrieben. Sie müssen keinerlei Wissen über das Programmieren mitbringen, um das Tutorial durchzuarbeiten.

Wenn Sie schon über Programmiererfahrung verfügen, sollten Sie dieses Tutorial überfliegen, um sich einen Überblick über die IDE (Integrated Development Environment, integrierte Entwicklungsumgebung) von REALbasic und über die Sprachfunktionen zu verschaffen.

Weitere Hinweise

Kursive Schrift wird verwendet, wenn ein Ausdruck zum ersten Mal gebraucht wird und um wichtige Konzepte hervorzuheben. Kursive Schrift wird außerdem für Buchtitel, wie z.B *REALbasic Entwicklerhandbuch*, eingesetzt.

Manchmal müssen Sie im Laufe des Tutorials Code-Zeilen in den Code-Editor eintippen. Diese werden mit einem unproportionalen Font hervorgehoben:

```
for i=1 to 10
   a(i)=i
next
```

Wenn Sie Code eintippen, beachten Sie bitte folgende Regeln:

- Tippen Sie jede abgedruckte Zeile in eine eigene Zeile im Code-Editor. Versuchen Sie nicht, zwei oder mehr Zeilen zu vereinigen oder eine lange Zeile auf zwei oder mehr Zeilen zu verteilen.
- Fügen Sie keine extra Leerzeichen ein, wo sich keine befinden.

Immer wenn Sie Ihr Programm starten, wird REALbasic zuerst die Syntax Ihres Programmtextes überprüfen. Die Syntaxüberprüfung wird Sie direkt auf die Zeile hinweisen, in der ein Problem auftritt. Vergleichen Sie diese Zeile mit der hier im Handbuch abgedruckten. Ebenso können Sie, falls es Probleme gibt, die zum Kapitel gehörige Quelltextdatei laden und den Code-Teil an entsprechender Stelle einfügen.

Tutorial-Dateien

Alle zum Tutorial zugehörigen Dateien finden Sie auf der CD im Ordner **Tutorial Projekt**. Sie können Ihre Arbeit an beliebiger Stelle im Tutorial mit den beigefügten Dateien vergleichen. Sie können ebenso ein neues Kapitel mit der fertigen Datei des vorigen Kapitels auf der CD beginnen.

Da für jedes Kapitel eine vollständige Projektdatei mitgeliefert wird, können Sie auch ein Kapitel überspringen, falls Sie darauf gerade keine Lust haben. Später können Sie das Ausgelassene immer noch aufarbeiten.

Deutsche Übersetzung

In diesem Handbuch wurde nicht mit Gewalt jeder Begriff ins Deutsche übersetzt. Englische Begriffe, die sich etabliert haben und für die es keine passende Übersetzung gibt, wurden so belassen (Code, Editor, Browser). Sie werden also nicht Flag durch Flagge und Key durch Schlüssel ersetzt finden.

Leider gibt es einige Stellen, an denen wir selbst unschlüssig waren, ob es nun besser ist, den englischen oder den deutschen Begriff zu verwenden. Meistens sprechen ebenso viele Argumente dafür wie dagegen. Wir bitten Sie daher, großzügig darüber hinwegzusehen, wenn Sie meinen, eine inkonsequente Haltung unsererseits entdeckt zu haben.

Auf die Plätze, fertig, los!

Sie sind nun bereit, mit dem Lernen von REALbasic zu beginnen.

2. Das Erzeugen von Fenstern

In diesem Kapitel erhalten Sie eine Einführung in REALbasic und seine integrierte Entwicklungsumgebung (IDE).

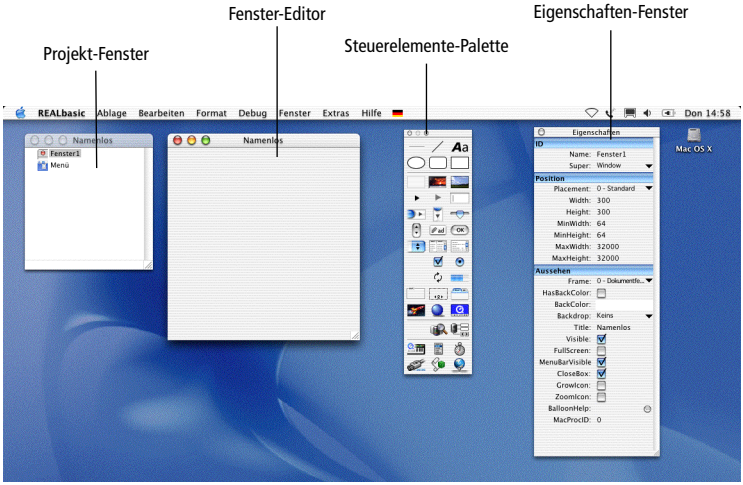
Sie werden erfahren,

- wie man REALbasic startet,
- welche Fenster zur Entwicklungsumgebung von REALbasic gehören,
- wie man ein Dokument-Fenster erzeugt, das den Text des Texteditors aufnehmen wird,
- wie man das selbstgeschriebene Programm startet.

REALbasic starten

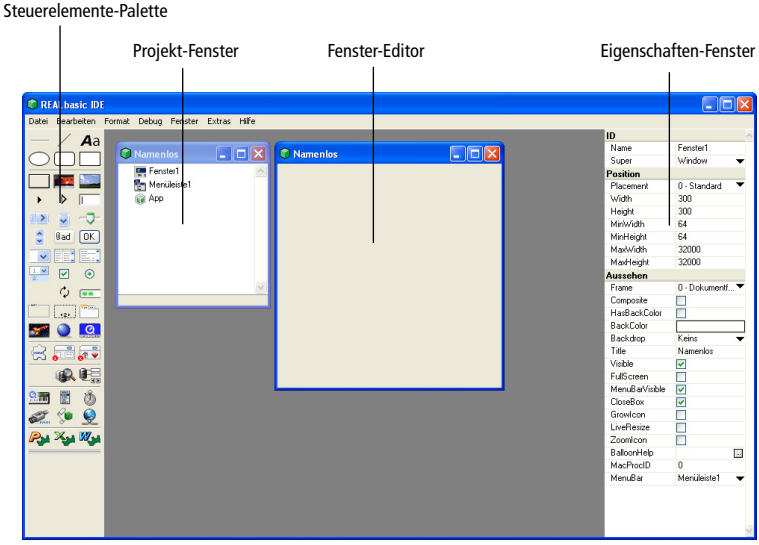
Suchen Sie das REALbasic-Programm-Icon auf Ihrem Desktop (es befindet sich in dem Ordner, in dem Sie REALbasic installiert haben) und starten Sie es durch Doppelklick.

Nachdem Sie REALbasic gestartet haben, sollte Ihr Bildschirm unter Mac OS X etwa so aussehen:



Wenn Sie eine ältere Mac OS-Version verwenden, wird Ihr Bildschirm ein wenig anders aussehen.

Falls Sie REALbasic für Windows verwenden, sieht der Bildschirm nach dem Start von REALbasic ungefähr so aus:



Die REALbasic-Fenster

Wie Sie im Bild sehen können, hat REALbasic nach dem Start vier Fenster geöffnet:

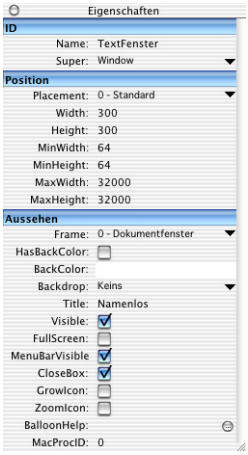
- Das *Projekt-Fenster* enthält eine Liste aller Teile Ihres REALbasic-Programms.
- *Fenster-Editor* ist der Oberbegriff für ein Applikationsfenster in der Entwicklungsumgebung. Den Namen eines speziellen Fensters finden Sie in der *Name*-Eigenschaft im *Eigenschaften-Fenster*.
- Die *Steuerelemente-Palette* enthält Icons, die Interface-Objekte repräsentieren, die Sie per Drag & Drop in den Fenster-Editor ziehen können, um die Interface-Objekte Ihrer Applikation zu erzeugen. Interface-Objekte werden in REALbasic als Steuerelemente bezeichnet.

- Das *Eigenschaften-Fenster* enthält eine Liste aller Eigenschaften und deren Werte für das *aktuell selektierte Objekt* Ihrer Applikation. Wenn Sie ein anderes Objekt selektieren, ändert sich das Eigenschaften-Fenster, um die Eigenschaften des neu selektierten Objektes darzustellen. Ist kein Objekt selektiert, bleibt das Eigenschaften-Fenster leer.

Zusätzlich können Sie folgende Fenster öffnen:

- Das *Farben-Fenster* wird benutzt, um selbstdefinierte Farben zu speichern, die Sie in Ihrem Programm verwenden möchten. Es besteht aus einer Palette von bis zu 16 Farben. Sie können einem Palettenelement eine Farbe zuweisen, indem Sie auf dieses klicken. Dann erscheint die Farbauswahl. Um einer Objekteigenschaft eine Farbe zuzuweisen, draggen Sie eine Farbe aus dem Farben-Fenster auf den Wert einer Eigenschaft, die eine Farbe akzeptiert, wie z.B. die BackColor-Eigenschaft eines Fenster-Objektes.
- Das *Sprachreferenz*-Fenster mit dem *Online-Referenzhandbuch*, das Sie über den Menüpunkt **Hilfe/Sprachreferenz** aufrufen. Dabei handelt es sich um eine bequeme Alternative zur gedruckten bzw. elektronischen (PDF-) Version der Sprachreferenz. Das Entwicklerhandbuch ist nicht Teil dieser Hilfe-Datei, Sie können sich aber die PDF-Version auf dem Bildschirm jederzeit anzeigen lassen.

Erzeugen eines Dokument-Fensters



Wenn Sie REALbasic starten, öffnet es zunächst ein leeres Fenster in einem Fenster-Editor. Der Name dieses Fensters ist **Fenster1** und wird im Projekt-Fenster aufgeführt, die Eigenschaften des Fensters werden im Eigenschaften-Fenster aufgelistet. Da dies das Fenster ist, das später den Texteditor enthalten wird, werden Sie ihm zunächst einen aussagekräftigeren Namen geben:

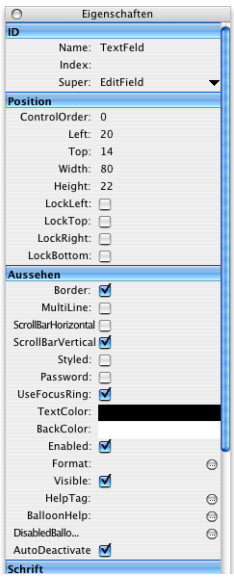
1. Klicken Sie auf **Fenster1** im Projektfenster. Das Eigenschaften-Fenster zeigt daraufhin die aktuellen Eigenschaften von Fenster1.
2. Ändern Sie die Eigenschaft **Name** im Eigenschaften-Fenster in **TextFenster**. Selektieren Sie dazu mit der Maus den Begriff **Fenster1**, tippen Sie `TextFenster` und drücken Sie **Return**.
   Dabei ändert sich auch im Projekt-Fenster der Name von **Fenster1** in **TextFenster**.
   Als nächstes müssen Sie dafür sorgen, dass man das Fenster später auch in der Größe ändern kann.

3. Aktivieren Sie dazu die Checkbox hinter **GrowIcon** im Eigenschaften-Fenster:

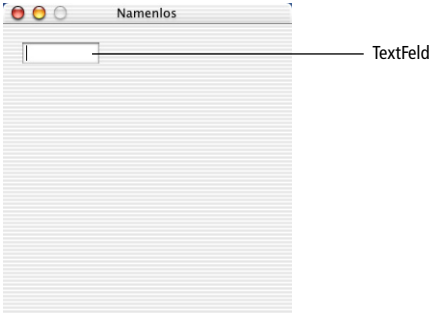
Hinzufügen eines EditField-Objektes

Damit unser TextFenster auch Text aufnehmen kann, benutzen Sie ein Steuerelement namens *EditField* aus der Steuerelemente-Palette. Das ist das Interface-Objekt, das Texteingaben vom Anwender entgegennimmt. Das **EditField**-Steuerelement sieht folgendermaßen aus:

Um ein EditField in das TextFenster einzufügen, müssen Sie folgendermaßen vorgehen:



1. Wenn das TextFenster noch nicht im Fenster-Editor geöffnet ist, öffnen Sie es mit einem Doppelklick im Projekt-Fenster.
2. Nehmen Sie das EditField in der Steuerelemente-Palette und ziehen Sie es per Drag & Drop in unser TextFenster. Da das EditField nun das aktuell selektierte Objekt ist, werden im Eigenschaften-Fenster seine Eigenschaften angezeigt.
   Sie verwenden die Steuerelemente der Steuerelemente-Palette als Vorlage für Ihre Interface-Objekte. Wenn Sie ein Steuerelement aus der Steuerelemente-Palette in ein Fenster ziehen, erzeugt REALbasic ein Objekt, das auf dieser Vorlage basiert. Dieses Objekt bekommt automatisch all die Eigenschaften, die zu der Vorlage gehören. Diese Eigenschaften werden im Eigenschaften-Fenster angezeigt.
3. Benutzen Sie das Eigenschaften-Fenster, um die Eigenschaft **Name** des EditField-Objektes von **EditField1** in **TextFeld** zu ändern.
4. Das Eigenschaften-Fenster des EditField-Objektes sollt nun so aussehen, wie nebenan gezeigt.
   Sie haben zwar erst damit begonnen, Ihre erste REALbasic-Applikation zu programmieren, wollen diese aber bestimmt schon mal starten, um zu sehen, was passiert.
5. Wählen Sie den Menüpunkt **Debug/Programm starten**. Das TextFenster erscheint und sollte etwa so aussehen:



6. Tippen Sie etwas in das TextFeld, um es zu testen.
7. Nachdem Sie alles ausprobiert haben, wählen Sie **Mein Programm/Beenden** unter Mac OS X oder **Datei/Beenden**, um zur Entwicklungsumgebung zurückzukehren.

Wenn Sie **Run** wählen, um Ihr Programm zu starten, schaltet REALbasic automatisch in die Runtime-Umgebung um (vorausgesetzt, es werden keine Syntaxfehler gefunden). Die Runtime-Umgebung verwenden Sie, um Testläufe Ihres Pro-

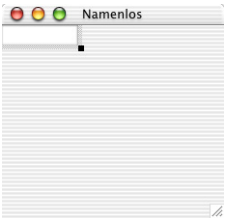
gramms durchzuführen und für das Debugging (die Fehlersuche). Nachdem Sie Ihr Programm beendet haben, landen Sie wieder in der Entwicklungsumgebung.

Das TextFeld zum Text-Editor umfunktionieren

Das TextFeld, das Sie gerade erzeugt haben, ist offensichtlich noch kein adäquater Texteditor. Bis jetzt kann das TextFeld nur eine kleine Menge Text verarbeiten und das auch nur in einer einzelnen Zeile. Um einen benutzbaren Editor zu bekommen, benötigt das TextFeld einen Scrollbalken und muss mehrere Zeilen Text akzeptieren. In diesem Abschnitt konfigurieren Sie das TextFeld so, dass es wie ein Texteditor funktioniert und vergrößern es so, dass seine Ausmaße das Fenster ausfüllen.

Um die linke obere Ecke des TextFeldes an der linken oberen Ecke des TextFensters zu fixieren und das EditField so zu vergrößern, dass es mehrere Zeilen Text aufnehmen kann, gehen Sie wie folgt vor:

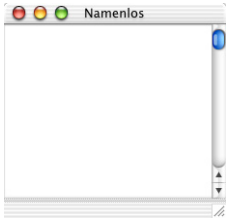
1. Klicken Sie auf das TextFeld, um es zu selektieren, falls es noch nicht selektiert ist. Sie sehen rechts im Eigenschaften-Fenster, dass der Name des selektierten Objekts TextFeld ist (wichtig!).
2. Suchen Sie im Eigenschaften-Fenster die **Top**- und **Left**-Eigenschaften (oben bzw. links). Selektieren Sie den Wert, der für die Left-Eigenschaft zuständig ist und tippen Sie dort `-1` ein (der ursprüngliche Wert verschwindet dadurch). Drücken Sie die Return-Taste, um der Eigenschaft den neuen Wert zuzuweisen.
   Dadurch wird das TextFeld an die linke Seite des TextFensters bewegt. Der Wert -1 setzt die linke Ecke des EditField-Objektes genau außerhalb des Rahmens des TextFensters. Im Eigenschaften-Fenster werden negative Werte zur besseren Erkennung in rot dargestellt.
3. Wiederholen Sie Schritt 2 für die Top-Eigenschaft. Ändern Sie auch diesen Wert auf -1.
   Das TextFeld ist nun mit dem oberen und linken Rand des Fensters ausgerichtet:



4. Draggen Sie nun die rechte untere Ecke des TextFeldes (den kleinen schwarzen Punkt) so weit nach rechts unten, dass es an den Vergrößerungsknopf des TextFensters anstößt.
5. Um die rechte Seite des TextFeldes an die rechte Seite des TextFensters anzugleichen, draggen Sie den Vergrößerungsknopf des TextFensters an die entsprechende Position. Das TextFenster sollte nun folgendermaßen aussehen:

   Als nächstes müssen Sie REALbasic mitteilen, dass das TextFeld so viele Textzeilen akzeptieren soll, wie der Anwender eingibt. Ebenso soll ein Scrollbalken angezeigt und der Text immer beim Erreichen des rechten Randes umgebrochen werden. Dies erreichen Sie ganz einfach durch Aktivieren der **MultiLine**-Eigenschaft für das TextFeld.
6. Selektieren Sie die **MultiLine**-Eigenschaft von TextFeld, indem Sie die entsprechende Checkbox im Eigenschaften-Fenster aktivieren.

Das TextFeld hat nun einen Scrollbalken. Das Dokumentfenster sollte nun so aussehen:



Um das Programm erneut zu starten, gehen Sie so vor:

1. Wählen Sie **Debug/Programm starten**. Das TextFenster-Fenster erscheint.
2. Geben Sie ein paar Zeilen Text ein.
   Während Sie tippen, werden Sie bemerken, dass der Text am Ende der Zeile umgebrochen wird. Sobald Sie genug getippt haben, um alle Zeilen des Fensters auszufüllen, wird der Scrollbalken aktiv. Sie können den Scrollbalken benutzen, um wieder zurück an den Anfang zu gelangen.
3. Wenn Sie fertig getippt haben, wählen Sie **Mein Programm/Beenden** unter Mac OS X oder **Datei/Beenden**, um zur Entwicklungsumgebung zurückzukehren.
4. Sie sollten Ihr Projekt jetzt speichern. Wählen Sie hierzu **Ablage/Sichern** (Windows: **Datei/Sichern**). Speichern Sie Ihre Projektdatei unter dem Namen **TextEditor-Kapitel2**. Der Titel des Projekt-Fensters ist nun „Text-Editor-Kapitel2".

Sollte Ihr Rechner einmal beim Testen Ihres Programms abstürzen, restauriert REALbasic den ursprünglichen Status Ihres Projektes, wenn Sie das Projekt öffnen. Sie müssen also Änderungen nicht andauernd speichern, um zu verhindern, dass Sie Ihre Arbeit verlieren.

Schließlich müssen Sie das TextFeld so konfigurieren, dass es die gleiche Größe wie das Fenster annimmt, wenn der Benutzer die Fenstergröße ändert. Wenn Sie dies nicht machen, haben TextFeld und das Fenster nur dann die gleiche Größe, wenn der Benutzer die Fenstergröße niemals ändert (und das ist ziemlich unwahrscheinlich).

Sie können dies austesten, indem Sie in den Laufzeitumgebung wechseln und das Fenster in der Größe ändern. Das sieht dann etwa folgendermaßen aus:

Das TextFeld behält die Größe bei, die Sie im Eigenschaften-Fenster festgelegt haben, das Fenster wurde in der Größe jedoch vom Anwender verändert. Jeder Anwender geht eigentlich davon aus, dass der Eingabebereich die gleiche Größe besitzt wie das Fenster.

REALbasic bietet dafür eine einfache Möglichkeit: Die Lock-Eigenschaften (Lock steht für Einrasten bzw. Ankoppeln). Die Lock-Eigenschaften werden verwendet, um den Abstand der TextFeld- und Fensterecken fest zu definieren. Dieser Abstand wird auch bei Größenänderungen des Fensters beibehalten, wenn die Lock-Eigenschaften aktiviert sind.

Um die Größe von TextFeld mit der des Fensters zu verbinden, gehen Sie folgendermaßen vor:

1. Selektieren Sie das TextFeld. Suchen Sie nach den Eigenschaften **LockLeft**, **LockTop**, **LockRight** und **LockBottom** im Eigenschaften-Fenster und aktivieren Sie diese durch Anklicken der entsprechenden Checkboxen.
2. Starten Sie Ihr Programm, um die Vergrößerungsfunktion zu testen.
3. Beenden Sie Ihr Programm, um zur Entwicklungsumgebung zurückzukehren und speichern Sie Ihr Projekt.

Bis hierher haben Sie ein sehr nützliches REALbasic-Objekt erzeugt: TextFenster. TextFenster enthält mit TextFeld ein Objekt, das von der EditField-Klasse abgeleitet und für Textbearbeitung konfiguriert wurde. TextFeld hat automatisch alle Eigenschaften und Methoden der EditField-Klasse geerbt. Es wurde so konfiguriert, dass es mehrzeilige Texte akzeptiert, über einen vertikalen Scrollbalken verfügt und seine Größe an die des umgebenden Fensters anpasst. Wenn Sie eine weitere Instanz von TextFenster erzeugen, erhalten Sie alle Eigenschaften von TextFeld „kostenlos" dazu. Diese Tatsache werden Sie ausnutzen, wenn Sie im 4. Kapitel den Menüpunkt **Ablage/Neu** (Windows: **Datei/Neu**) implementieren.

Der einfachste Weg, das TextFenster in einem neuen Projekt wiederzuverwenden besteht darin, es aus dem Projekt-Fenster auf den Desktop (oder in jedes andere Verzeichnis) zu ziehen. Diese Aktion speichert es als exportiertes REALbasic-Objekt. Wenn Sie es in einer anderen Anwendung wiederverwenden möchten, dann ziehen Sie es einfach vom Finder in das Projekt-Fenster des anderen Projekts.

Rückblick

In diesem Kapitel haben Sie erfahren, wie man REALbasic startet, welche Fenster es in der Entwicklungsumgebung gibt und wie man ein mehrzeiliges EditField in ein Fenster platziert und die eigene Applikation startet.

3. Erzeugen von Menüeinträgen

In diesem Kapitel werden Sie mit Menüs in REALbasic konfrontiert. Sie werden lernen, wie man

- einer Applikation Menüeinträge hinzufügt,
- einen Menüeintrag aktiviert.

Zum Ausprobieren der Beispiele können Sie an der Applikation weiterarbeiten, die Sie im 2. Kapitel begonnen haben oder die Datei **TextEditor-Kapitel2** laden, die Sie im Ordner „Tutorial Projekt" auf Ihrer CD finden.

Hinzufügen eines Alles auswählen-Menüpunkts

In dieser Übung werden Sie dem **Bearbeiten**-Menü einen Menüpunkt **Alles auswählen** hinzufügen. Hierzu sind folgende drei Schritte notwendig:

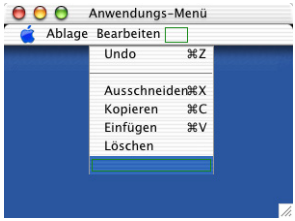
1. Einfügen des Menüeintrags in das Bearbeiten-Menü.
2. Hinzufügen eines *Menu-Handlers*, der festlegt, wie REALbasic reagieren soll, wenn der Anwender ein (aktives) Menü anwählt. Der Menu-Handler kann weitere Methoden (Prozesse) aufrufen.

Hinzufügen des Menüeintrags

Um einen Menüeintrag **Alles auswählen** hinzuzufügen, gehen Sie folgendermaßen vor:

1. Holen Sie das Projekt-Fenster nach vorne (z.B. durch **Fenster/Projekt**).
2. Führen Sie einen Doppelklick auf dem Menü-Objekt aus, um den Menü-Editor zu öffnen.
3. Selektieren Sie das **Bearbeiten**-Menü im Menü-Objekt-Fenster. Das Menü klappt herunter.

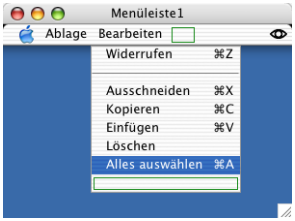
4. Selektieren Sie den leeren Menüeintrag am Ende der Liste:



   Im Menü-Editor ist immer am Ende eines jeden Menüs ein leerer Eintrag. Sie können diesen verwenden, um neue Menüeinträge zu erzeugen. Der leere Eintrag ist eigentlich kein Bestandteil des Menüs und erscheint auch später nicht in der fertigen Applikation.

   Wenn Sie versehentlich einen Menüeintrag hinzugefügt haben, können Sie ihn entfernen, indem Sie ihn selektieren und Backspace drücken.

5. Im Eigenschaften-Fenster geben Sie als **Text**-Eigenschaft **Alles auswählen** ein.
6. Die **Name**-Eigenschaft wird automatisch mit **BearbeitenAllesauswhlen** ausgefüllt. Dies ändern Sie in **Bearbeiten-Allesauswaehlen**, da Menünamen keine Sonderzeichen enthalten dürfen. Ebenso sind Leerzeichen nicht erlaubt.
7. Weisen Sie der **CommandKey**-Eigenschaft den Buchstaben **A** zu.
8. Deselektieren Sie den Schalter **AutoEnable**. Dieser würde bewirken, dass der Menüpunkt automatisch immer auswählbar ist. Da er aber nur dann auswählbar sein soll, wenn ein Textfenster offen ist, müssen wir den Menüpunkt an anderer Stelle aktivieren.
9. Der Menü-Editor und das Eigenschaften-Fenster sollten nun wie folgt aussehen:



   Das Eigenschaften-Fenster zeigt, dass die übergeordnete Klasse des Objekts (Super Class – die Klasse, auf der das Objekt basiert) „MenuItem“ ist. Das bedeutet, dass es alle Eigenschaften dieser Klasse erbt und Sie die Eigenschaften und Methoden der MenuItem-Klasse verwenden können, um den „Alles auswählen“-Menüeintrag zu verwalten.

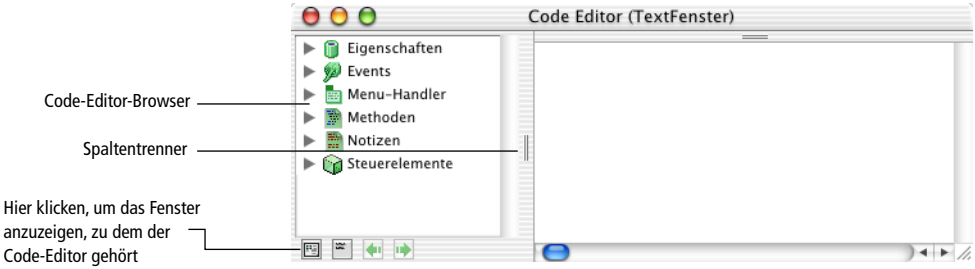
10. Schließen Sie den Menü-Editor.

Aktivieren des Menüeintrags

Da der Menüeintrag „Alles auswählen“ mit dem TextFeld in TextFenster in Verbindung steht, fügen Sie den Code, der den Menüeintrag zu aktiviert, in das TextFenster-Objekt ein. Es gibt keinen Grund, den Menüeintrag zu aktivieren, wenn kein Fenster offen ist oder kein selektierbarer Text vorhanden ist.

Die MenuItem-Klasse besitzt eine Methode namens **Enable**, welche die Enable-Eigenschaft des Menüeintrags auf **True** setzt. Dies aktiviert den Menüeintrag „Alles auswählen“ jedesmal, wenn der Anwender ins Bearbeiten-Menü klickt, um die dort vorhandenen Menüeinträge anzuzeigen. Um den Menüeintrag „Alles auswählen“ zu aktivieren, gehen Sie folgendermaßen vor:

1. Selektieren Sie **TextFenster** im Projekt-Fenster und drücken Sie **alt-Tab**, um den Code-Editor zu öffnen.



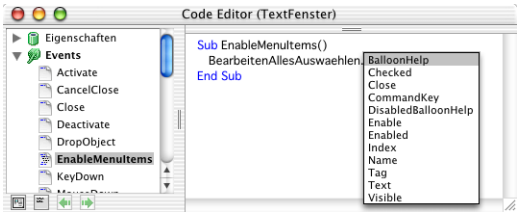
   Im Browser, dem linken Teil des Code-Editor-Fensters, können Sie die einzelnen Einträge durch Klick auf das Dreieck vor dem Namen eines Eintrags expandieren und den verschiedenen Events oder Objekten Code hinzufügen. Außerdem können Sie den Browser-Bereich durch Ziehen des Spaltentrenners nach rechts oder links vergrößern und verkleinern.

2. Klicken Sie im Browser auf das Dreieck links vom **Events**-Icon und selektieren Sie **EnableMenuItems** (eventuell müssen Sie den Spaltentrenner nach rechts ziehen, um die Namen aller Events lesen zu können oder den Scrollbalken benutzen). Dabei handelt es sich um einen Event-Handler, der immer dann ausgeführt wird, wenn sich der Anwender die Einträge eines Menüs anzeigen lässt.

3. Beginnen Sie, folgende Zeile in den Code-Editor zu tippen:
   `BearbeitenAllesauswaehlen.Enable`
   Hinweis: Während Sie tippen, versucht REALbasic zu erraten, was Sie eintippen wollen. Wenn es einen Vorschlag hat, zeigt es diesen hellgrau an. Sobald Sie die Buchstaben „Bea“ eingegeben haben, vermutet es, dass Sie „Bearbeiten“ tippen wollen.

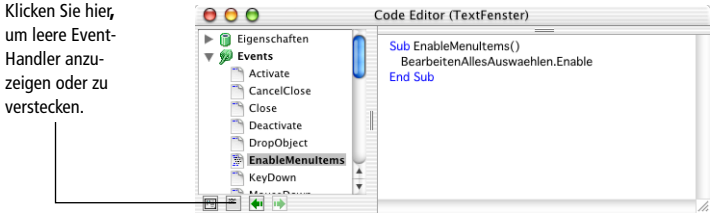
4. Drücken Sie die Tab-Taste, um den Vorschlag zu bestätigen. Der Code-Editor fügt an der Einfügemarke „Bearbeiten“ ein. Vervollständigen Sie den Text zu `BearbeitenAllesauswaehlen`.

5. Tippen Sie einen Punkt, der als Trenner zwischen Objektnamen und seiner Eigenschaft verwendet wird.
   Der Code-Editor zeigt drei Punkte an. Wenn Sie darauf die Tab-Taste drücken, erhalten Sie ein Kontextmenü mit allen Eigenschaften, die zu Objekten des Typs MenuItem gehören, wie in folgender Abbildung zu sehen:



   Wenn Sie den Buchstaben „e“ nach dem Punkt tippen, schlägt REALbasic „Enable“ vor und Sie können Tab drücken, um das zu übernehmen. Alternativ können Sie sich mit den Cursortasten hoch/runter bis zu „Enable“ vorarbeiten.

Wenn der Code-Editor Objekt- oder Eigenschaften-Namen vorschlägt, können Sie diese durch Weitertippen verwerfen. Sie übernehmen einen bestimmten Vorschlag oder rufen das Kontextmenü mit der Tab-Taste auf. Der Code-Editor von TextFenster sollte nun so aussehen:



Der Event „EnableMenuItems“ wird nun fett dargestellt, um anzuzeigen, dass er Code enthält. Das zweite Icon unter dem Browser-Bereich erlaubt Ihnen, entweder alle Events anzuzeigen oder nur die, die Code enthalten. Normalerweise werden alle Events angezeigt, damit Sie, wenn Sie ein Programm schreiben, Code in die leeren Event-Handler hineinschreiben können. Später werden Sie es begrüßen, leere Events zur besseren Übersicht ausblenden zu können.

Speichern Sie Ihr Projekt unter dem Namen **TextEditor-Kapitel3**.

Einem Menüeintrag eine Funktion zuweisen

Als letztes müssen Sie REALbasic mitteilen, was passieren soll, wenn der Menüeintrag **Alles auswählen** angewählt wird. Dies nennt man einen *Menu-Handler*. Der Menü-Handler läuft automatisch ab, wenn der Anwender den Menüpunkt **Alles auswählen** aufruft.

In diesem Fall führt der Menü-Handler die Textselektion durch.

Folgendermaßen programmieren Sie einen Menu-Handler für den Menüeintrag „Alles auswählen“:

1. Wählen Sie, wenn der Code-Editor für TextFenster das oberste Fenster ist, den Menüpunkt **Bearbeiten/Neuer Menu-Handler**.
2. Wählen Sie **BearbeitenAllesauswaehlen** aus dem Popup-Menü und klicken Sie auf **OK**.

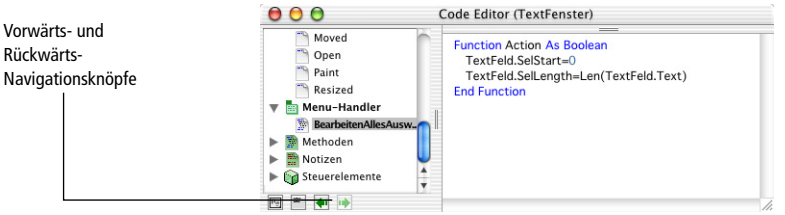
Eine neue Methode namens **BearbeitenAllesauswaehlen** erscheint in der Menu-Handlers-Kategorie des Code-Editors von TextFenster. Der Code-Editor zeigt die Funktionsdeklaration.

3. Tippen Sie folgendes ein:

```
TextFeld.SelStart=0
TextFeld.SelLength=Len(TextFeld.Text)
```

Dieser Programmcode verwendet zwei Eigenschaften des EditField-Objekts – SelStart und SelLength – um zu bestimmen, welcher Text selektiert werden soll. SelStart legt die Position des ersten hervorzuhebenden Zeichens und SelLength die Länge der Selektion fest, beginnend bei SelStart. Die Length-Funktion ist eine globale Funktion, die die Länge der an sie übergebenen Zeichenkette zurückliefert. In unserem Fall wird der gesamte Text des TextFelds an die Length-Funktion übergeben.

Diese Zeilen setzen den Beginn der Selektion auf den Beginn des Textes und die Länge der Selektion auf die Länge des Textes im TextFeld (die Text-Eigenschaft eines EditField enthält den Text des EditFields).
Der Code-Editor von TextFenster sollte nun so aussehen:



Nachdem Sie den Code für einen zweiten Event-Handler eingegeben haben, ist der linke Pfeil unter dem Browser-Bereich aktiviert. Die Pfeile sind Navigationsknöpfe und funktionieren wie die Navigationsknöpfe in einem Web-Browser. Sie können zum EnableMenuItems-Event springen, indem Sie auf den Rückwärts-Pfeil klicken und wieder zurück zum Menu-Handler, indem Sie dann auf den Vorwärts-Pfeil klicken.

4. Speichern Sie Ihr Projekt. Stellen Sie sicher, dass der Name **TextEditor-Kapitel3** lautet.
5. Starten Sie Ihr Programm. Falls dabei Probleme beim Compilieren auftreten, überprüfen Sie bitte, ob Sie das EditField in TextFeld umbenannt haben. Falls Sie diesen Schritt versehentlich übersprungen haben, kann REALbasic Referenzen auf Eigenschaften von TextFeld nicht auflösen.
6. Tippen Sie ein wenig Text in den Texteditor und testen Sie den Menüpunkt **Alles auswählen**. Sie sollten sowohl über den Menüpunkt als auch über den Tastatur-Shortcut Kommando-A in der Lage sein, den gesamten Text zu selektieren, wie in folgender Abbildung gezeigt:

7. Wenn Sie fertig sind, beenden Sie Ihr Programm und kehren zur Entwicklungsumgebung zurück.

Rückblick

In diesem Kapitel haben Sie erfahren, wie man Menü-Einträge hinzufügt, diese aktiviert (enabled) und wie deren Ereignisse (Events) bearbeitet werden.

4. Arbeiten mit Dokumenten

In diesem Kapitel werden Sie mit Dokumenten arbeiten. Sie werden lernen, wie man:

- Menüeinträge erzeugt, um Dokumente zu erzeugen, zu öffnen und zu speichern,
- Programmcode schreibt, um diese Menüeinträge abzuarbeiten,
- eine „Änderungen sichern"-Dialogbox implementiert, die erscheint, wenn der Benutzer das Dokumentfenster schließt oder das Programm beenden will, obwohl es noch ungesicherte Änderungen gibt.

Los geht's

Suchen Sie die REALbasic-Projektdatei, die Sie am Ende des letzten Kapitels gespeichert haben (**TextEditor-Kapitel3**). Starten Sie REALbasic und öffnen Sie das Projekt. Falls nötig, verwenden Sie die mitgelieferte Datei **TextEditor-Kapitel3**, die Sie im Ordner **Tutorial Projekt** finden.

Arbeiten mit Textdokumenten

Ein Texteditor muss in der Lage sein, Textdokumente zu laden und zu speichern. Als erstes werden Sie das Projekt um eine Funktion zum Anlegen neuer Dokumente erweitern. Wie Sie im vorigen Kapitel erfahren haben, sind drei Schritte erforderlich, um einen neuen Menüeintrag zu implementieren:

- Hinzufügen eines Menüeintrags in die Menüzeile,
- Hinzufügen eines Menu-Handlers.

Erzeugen des Menüeintrags Neu

Um den **Neu**-Menüeintrag zu erzeugen, gehen Sie folgendermaßen vor:

1. Führen Sie im Projekt-Fenster einen Doppelklick auf das Menü-Objekt aus und selektieren Sie den leeren Menüeintrag im Menü **Ablage** (Windows: **Datei**).
2. Im Eigenschaften-Fenster geben Sie für die **Text**-Eigenschaft **Neu** und für die **CommandKey**-Eigenschaft **N** ein.

   Tragen Sie als Name-Eigenschaft **AblageNeu** ein.
3. Draggen Sie den Menüeintrag **Neu** an den Anfang des Menüs.

   Ihr Menü-Editor und das Eigenschaften-Fenster des Menüeintrags **Neu** sollten jetzt folgendermaßen aussehen:

4. Schließen Sie den Menü-Editor.

Verwalten des Menüeintrags Neu

Jetzt müssen Sie einen Menu-Handler schreiben, der ausgeführt wird, wenn der Benutzer den Menüeintrag aufruft. Ohne einen passenden Menu-Handler würde der Menüeintrag gar nichts bewirken.

Der Menu-Handler für den Menüpunkt **Neu** erzeugt ein neues Objekt des Typs **TextFenster**. Dabei handelt es sich um eine Instanz der Objektklasse TextFenster mit all den in Kapitel 2 vereinbarten Eigenschaften. TextFenster ist ein Fenster, das bereits ein als Texteditor konfiguriertes TextFeld-Objekt enthält.

Um den Menüeintrag **Neu** zu verwalten, ist folgendes zu tun:

1. Stellen Sie sicher, dass der Code-Editor von **App** geöffnet und das aktive Fenster ist. Ist dies nicht der Fall, müssen Sie **App** im Projektfenster aktivieren und **alt-Tab** drücken. Rufen Sie den Menüpunkt **Bearbeiten/Neuer Menu-Handler** auf, um einen Menu-Handler für den Menüpunkt **Neu** zu erzeugen.
2. Wählen Sie den Menu-Handler **AblageNeu** aus und klicken Sie auf OK.
3. Geben Sie für den Menu-Handler folgenden Code in den Code-Editor ein:

```
Dim w as TextFenster
w=New TextFenster
```

   Die Dim-Anweisung erzeugt eine neue Variable des Typs **TextFenster**, nicht aber eine Instanz von **TextFenster**. Die Instanz wird erst in der nächsten Zeile durch den **New**-Operator kreiert und in **w** zurückgeliefert. Die neue Instanz

w ist ein Klon von **TextFenster** und wird sofort angezeigt, da Sie im 2. Kapitel die **Visible**-Eigenschaft aktiviert hatten.

4. Speichern Sie Ihr Projekt unter **TextEditor-Kapitel4** und wechseln Sie dann zur Runtime-Umgebung, um den Menüeintrag **Neu** zu testen. Sie werden feststellen, dass durch Aufrufen von **Ablage/Neu** (Windows: **Datei/Neu**) ein Klon des originalen **TextFensters** erzeugt wird. Das funktioniert auch dann, wenn gar kein **TextFenster** offen ist.

   Beenden Sie Ihr Programm, um in die Entwicklungsumgebung zurückzukehren.

Dateitypen

Sie können festlegen, welche Dateitypen Ihr Programm erkennen soll. Wenn Sie zum Beispiel ein Grafikprogramm schreiben möchten, müssen Sie REALbasic mitteilen, dass es Dateien der Typen PICT, TIFF, etc. laden können soll. Falls Ihr Programm beim Speichern ein eigenes Dateiformat erzeugt, muss REALbasic dessen Dateityp ebenfalls kennen.

Man verwendet die **Dateitypen**-Dialogbox, um die erlaubten Dateitypen zu definieren. Sie können die **Dateitypen**-Dialogbox durch Anwahl von **Bearbeiten/Dateitypen** aufrufen. Die TextEditor-Applikation soll in der Lage sein, Textdateien zu öffnen, zu modifizieren und zu speichern. Auf dem Macintosh verwendet das Programm den Dateityp „TEXT“, unter Windows den Dateityp „RTF“ (Rich Text Format). Während der Dateityp TEXT standardmäßig unterstützt wird, müssen Sie RTF noch hinzufügen.

Den Dateityp RTF hinzufügen

1. Wählen Sie den Menüpunkt **Bearbeiten/Dateitypen**. Es erscheint die Dateitypen-Dialogbox. Der Dateityp TEXT ist bereits definiert.

2. Klicken Sie auf den Knopf **Neu...**, um den Dateitypen-Editor aufzurufen.
3. Nennen Sie den Dateityp **rtf**, geben Sie als Mac-Creator **????**, als Mac-Typ **TEXT** und als Extension **.rtf** ein, wie in der folgenden Abbildung gezeigt. Creator und Typ sind case-sensitiv (es wird zwischen Groß- und Kleinschreibung unterschieden).

   Der Creator-Code **????** sorgt dafür, dass unser TextEditor Text-Dateien öffnet, die von einem beliebigen Programm erzeugt wurden. Wenn Sie einen bestimmten Creator-Code angeben – wie z.B. „ttxt“ oder „R*ch“ – wird der TextEditor nur solche Textdateien öffnen können.

4. Klicken Sie auf **OK**, um den RTF-Dateityp abzuspeichern und im Dateitypen-Dialog auf **OK**, um diesen zu schließen.

Speichern von Dokumenten

Da Dokumente nur dann gespeichert werden können, wenn es ein offenes Dokumentfenster gibt, übertragen Sie diese Aufgabe dem TextEditor-Fenster. Fügen Sie dazu dem „TextFenster“ den Menüpunkt **Sichern** hinzu. Dabei ist Folgendes zu beachten:

- Der Menüpunkt **Sichern** sollte nur dann aktiviert sein, wenn sich der Inhalt des aktuellen Fensters geändert hat – und nicht ständig, wie dies für die Menüeinträge **Neu** und **Alles auswählen** der Fall ist.
- Erzeugen eines Menu-Handlers: Der Menu-Handler für den Menüpunkt **Sichern** wird eine eigene Methode aufrufen, in der die Datei gespeichert wird.
- Hinzufügen der Methode zum Speichern der Datei: Diese Methode verwendet eine Objekt-Klasse namens **FolderItem**, um den Inhalt des Fensters in einer Textdatei auf der Festplatte zu speichern.

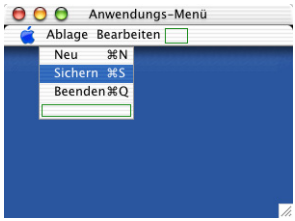
Hinzufügen des Menüpunkts „Sichern“

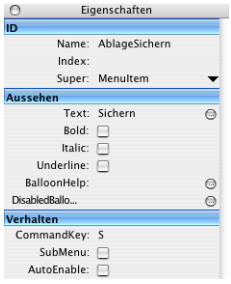
Um den Menüpunkt **Sichern** hinzuzufügen, gehen Sie folgendermaßen vor:

1. Bringen Sie den Menü-Editor nach oben oder führen Sie, falls dieser noch nicht geöffnet ist, einen Doppelklick auf das Menü-Objekt im Projekt-Fenster aus.
2. Selektieren Sie den leeren Menüeintrag im Menü **Ablage** (Windows: **Datei**) und verwenden Sie das Eigenschaften-Fenster, um ihm den Text **Sichern** und den CommandKey **S** zuzuweisen.

   Ändern Sie – falls notwendig – die Name-Eigenschaft in **AblageSichern**.
3. Deselektieren Sie unter **Verhalten** die **AutoEnable**-Eigenschaft.

   Wenn AutoEnable aktiv ist, ist der Menüpunkt immer anwählbar. Dies haben Sie sich bereits beim Menüpunkt **Neu** zu Nutze gemacht. Für das Speichern-Menü wäre dieses Verhalten nicht angebracht. Dieses soll weder aktiv sein, wenn gar kein Fenster geöffnet ist (trifft nur auf Mac OS zu), noch wenn es seit dem letzten Speichern keine Änderungen am Dokument gegeben hat.
4. Draggen Sie den Menüpunkt **Sichern** zwischen den **Neu**- und den **Beenden**-Eintrag.
   Der Menü-Editor sollte nun folgendermaßen aussehen:





5. Schließen Sie den Menü-Editor.

Hinzufügen neuer Eigenschaften zu TextFenster

Wenn Sie in einem Programm eine Datei laden, müssen Sie sich den Dateinamen merken, damit Sie Änderungen am Text später speichern können. Zu diesem Zweck definieren Sie eine neue Eigenschaft für TextFenster, die den Dateinamen aufnimmt. Dies ist deswegen sinnvoll, da jedes weitere TextFenster, das geöffnet wird, einer bestimmten Datei zugeordnet ist. Außerdem werden Sie eine Eigenschaft hinzufügen, die Änderungen am Text seit dem letzten Speichern mitverfolgt.

Um eine neue Eigenschaft zu **TextFenster** hinzuzufügen, gehen Sie folgendermaßen vor:

1. Selektieren Sie TextFenster im Projekt-Fenster und drücken Sie alt-Tab, um den Code-Editor zu öffnen.
2. Wählen Sie den Menüpunkt **Bearbeiten/Neue Eigenschaft**.
   Der Eigenschaften-Dialog erscheint.

3. Geben Sie `Dokument As FolderItem` ein, wählen Sie als Gültigkeitsbereich „Öffentlich (Überall)“ und klicken Sie dann auf **OK** (**FolderItem** ist der Name des REALbasic-Objekts, das auf Dateien und Ordner verweist).
4. Wählen Sie **Bearbeiten/Neue Eigenschaft**. Der Dialog zur Deklaration einer neuen Eigenschaft erscheint.
5. Geben Sie `TextWurdeGeändert As Boolean` ein, wählen Sie als Gültigkeitsbereich „Öffentlich (Überall)“ und klicken Sie dann auf **OK**. Der Datentyp Boolean kann genau zwei Werte annehmen: True (Wahr) oder False (Falsch). Im Abschnitt „Verwalten der TextWurdeGeändert-Eigenschaft“ auf Seite 27 werden Sie diese Eigenschaft verwenden, um festzustellen, ob sich der Inhalt von TextFeld geändert hat.
6. Klicken Sie auf das Dreieck links neben **Eigenschaften** im Code-Editor für **TextFenster**.
   Die Eigenschaften **Dokument** und **TextWurdeGeändert** werden aufgelistet. Der Eintrag **Eigenschaften** selbst ist fett geschrieben, um anzuzeigen, dass Eigenschaften hinzugefügt wurden.

Aktivieren der Menüeinträge

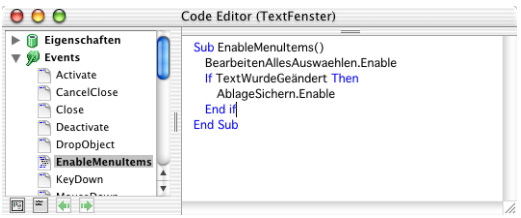
Da der Menüpunkt **Sichern** nur anwählbar sein soll, wenn es ungesicherte Änderungen im Dokument gibt, muss unser Quelltext die **TextWurdeGeändert**-Eigenschaft verwenden, die Sie gerade hinzugefügt haben.

Die **TextWurdeGeändert**-Eigenschaft dient als Markierung (Flag), an der REALbasic erkennen kann, ob der Anwender den Text im **TextFenster** verändert hat.

Um den Menüeintrag zu aktivieren, gehen Sie folgendermaßen vor:

1. Falls der Code-Editor von TextFenster noch nicht geöffnet ist, wählen Sie TextFenster im Projektfenster an und öffnen Sie den Code-Editor mit alt-Tab.
2. Klicken Sie auf das Dreieck links neben dem **Events**-Eintrag, um die Events anzuzeigen.
3. Selektieren Sie **EnableMenuItems** und fügen Sie folgende Zeilen am Ende der Methode an:

```
If TextWurdeGeändert Then
   AblageSichern.Enable
End if
```



4. Speichern Sie Ihr Projekt.

Hinzufügen einer DateiSichern-Methode

Als nächstes müssen Sie die DateiSichern-Methode implementieren, die ausgeführt wird, wenn der Anwender den Sichern-Menüpunkt aufruft. Diese Methode wird durch die Menu-Handler der Menüpunkte **Sichern** und **Sichern unter** aufgerufen. Den „Sichern unter"-Menüeintrag werden Sie später anlegen.

Dies DateiSichern-Methode wird zwei Fälle behandeln:

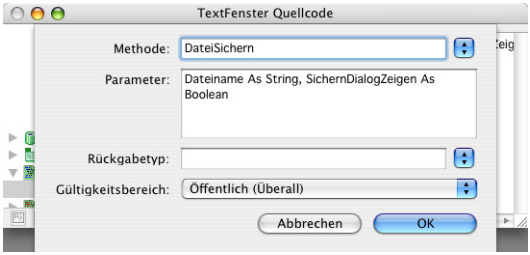
- Der Benutzer wählt **Sichern**, um ein existierendes Dokument zu sichern.
- Der Benutzer wählt **Sichern** oder **Sichern unter**, um ein ungesichertes oder ein bereits existierendes Dokument unter einem neuen Namen zu speichern.

Im letzten Fall muss das Programm zunächst eine Dateiauswahlbox anzeigen, damit der Benutzer einen Dateinamen eingeben kann. Im anderen Fall speichert das Programm das Dokument unter dem bestehenden Dateinamen.

Die Methode **DateiSichern** ruft abhängig vom Wert der Boole'schen Variablen **SichernDialogZeigen** die Dateiauswahlbox zur Eingabe des Dateinamens auf. Deshalb kann **DateiSichern** verwendet werden, um sowohl den Menüpunkt **Sichern**, als auch den Menüpunkt **Sichern unter** zu verarbeiten.

Um die **DateiSichern**-Methode hinzuzufügen, müssen Sie folgendermaßen vorgehen:

1. Sorgen Sie dafür, dass der Code-Editor für das TextFenster das oberste Fenster ist und klicken Sie auf **Bearbeiten/ Neue Methode**. Die Dialogbox für eine neue Methode erscheint.
2. Geben Sie der Methode den Namen `DateiSichern`.
   Im Parameter-Feld geben Sie folgendes ein: `Dateiname as String,SichernDialogZeigen As Boolean`
   Der „Neue Methode"-Dialog sollte jetzt folgendermaßen aussehen:



3. Klicken Sie auf **OK**, um die Dialogbox zu schließen.
   Der Code-Editor für die **DateiSichern**-Methode erscheint. Sie werden feststellen, dass der Methoden-Name und seine Parameter hinzugefügt wurden. Wenn Sie den Namen oder einen Parameter ändern möchten, können Sie dies nach einen Doppelklick auf die **DateiSichern**-Methode im Browser-Bereich des Code-Editors.
   Im nächsten Schritt geben Sie den Code ein, der die beiden beschriebenen Fälle behandelt.
4. Geben Sie folgenden Code für die **DateiSichern**-Methode in den Code-Editor ein:

```
Dim f as FolderItem
If Dokument=Nil or SichernDialogZeigen then
    f=GetSaveFolderItem("text",Dateiname)
    If f<>nil then //wenn der Anwender Sichern angeklickt hat
        Title=f.name
        Dokument=f
    End if
End if
If Not SichernDialogZeigen then //Anwender wählte Sichern
    If Dokument<>nil Then
        Dokument.SaveStyledEditField TextFeld
        TextWurdeGeändert=False
    End If
Elseif SichernDialogZeigen then //Anwender wählte SichernAls oder Neues Dokument
```

```
        If Dokument <> Nil and f <> Nil then
            Dokument.SaveStyledEditField TextFeld
            TextWurdeGeändert=False
        End if
    End if
```

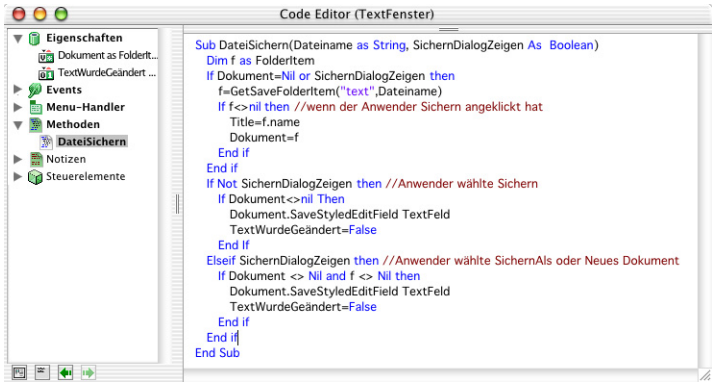
Jede gedruckte Zeile muss in eine separate Zeile in den Code-Editor eingegeben werden. Lange Zeilen dürfen nicht auf zwei Zeilen aufgeteilt werden.

Falls die **Dokument**-Eigenschaft nicht definiert ist (ihr Wert also „Nil“ ist), wurde das Dokument noch nicht gesichert, so dass die „Datei sichern unter“-Dialogbox angezeigt werden muss (dafür ist die **GetSaveFolderItem**-Funktion zuständig).

Die Zeile `Title=f.name` sorgt dafür, dass der Fenstertitel auf den Namen des FolderItem-Objekts – also des Dokuments – gesetzt wird. **Title** ist eine Eigenschaft der Fenster-Klasse, von der **TextFenster** abgeleitet wurde. Die Zeile `Dokument=f` weist der Dokument-Eigenschaft das aktuelle Dokument zu.

Falls das Dokument bereits existiert, seine Dokument-Eigenschaft also nicht „Nil“ ist, muss die „Datei sichern unter“-Dialogbox nicht gezeigt werden, da der Anwender das bereits existierende Dokument unter dem bestehenden Namen abspeichern will. In diesem Fall wird die **SaveStyledEditField**-Methode aufgerufen, um den Inhalt von TextFeld abzuspeichern. Dabei ist TextFeld der Wert des Parameters, der an die **SaveStyledEditField**-Methode der EditField-Klasse übergeben wird. Außerdem wird die Boole'sche **TextWurdeGeändert**-Eigenschaft wieder auf False gesetzt, da das Dokument seit dem letzten Sichern nicht mehr geändert wurde.

Der Code-Editor sollte nun so aussehen:



Bevor Sie die neue Methode verwenden können, muss allerdings noch die Code-Zeile eingefügt werden, die dafür sorgt, dass die TextWurdeGeändert-Eigenschaft auf True gesetzt wird, sobald der Anwender Änderungen am Text vornimmt. Darum werden Sie sich im Abschnitt „Verwalten der TextWurdeGeändert-Eigenschaft“ auf Seite 27 kümmern.

Verwenden der Online-Referenz

Die eben implementierte DateiSichern-Methode verwendet zwei in REALbasic integrierte Methoden, um die eigentliche Arbeit zu verrichten: Sie ruft die globale Methode **GetSaveFolderItem** auf, um die „Datei sichern“-Dialogbox darzustellen und macht von der **SaveStyledEditField**-Methode der FolderItem-Klasse Gebrauch, um den Inhalt des zum Text-Fenster gehörenden EditFields abzuspeichern. Wenn Sie es wünschen, können Sie diese Methoden in der gedruckten REALbasic Sprachreferenz nachschlagen oder bequemer in der Online-Version dieser Referenz nachschauen.

Um GetSaveFolderItem nachzuschlagen, gehen Sie folgendermaßen vor:

1. Rufen Sie den Menüpunkt Hilfe/Sprachreferenz auf. Damit öffnen Sie den Dialog der Online-Referenz. Der Browser im linken Teil des Dialogs listet alle Haupteinträge auf, und zwar wahlweise thematisch oder alphabetisch geordnet.

Wonach sortiert werden soll, können Sie durch Klick auf Alpha oder Theme festlegen. Im Kopfbereich gibt es ein Suchfeld, das Sie zur Suche nach Sprachelementen oder anderen Ausdrücken verwenden können.

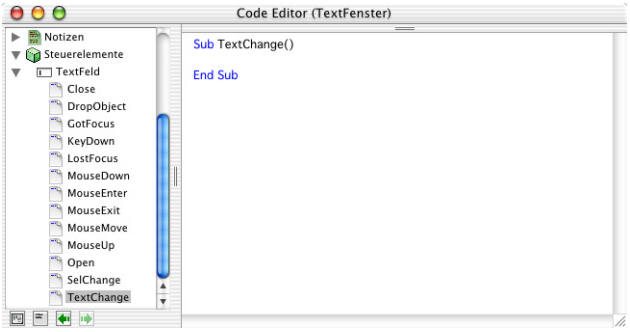
2. Geben Sie GetSaveFolderItem im Suchfeld ein und klicken Sie auf **Suchen**. Wie immer versucht REALbasic, beim Tippen das Wort zu erkennen, das Sie eingeben. Drücken Sie zu einem beliebigen Zeitpunkt Tab, um ein Kontextmenü aufzurufen oder den Vorschlag zu übernehmen.
3. Im Hauptbereich des Referenz-Fensters erscheint die Dokumentation zu GetSaveFolderItem. Dabei werden Hypertext-Verweise auf verwandte Themen blau und unterstrichen dargestellt. Sie können nun auf einen der FolderItem-Verweise klicken und dort in der Methodentabelle die SaveStyledEditField-Methode nachlesen. Die Pfeile im Kopfbereich sind Vorwärts- und Rückwärts-Knöpfe und funktionieren wie in einem Web-Browser.
   Code-Beispiele werden in gepunkteten Rahmen angezeigt und können per Drag & Drop in den Code-Editor übernommen werden.
4. Um die Online-Referenz wieder zu schließen, klicken Sie einfach auf den Schließknopf des Fensters.

Verwalten der TextWurdeGeändert-Eigenschaft

Der TextWurdeGeändert-Eigenschaft muss immer dann der Wert True („wahr") zugewiesen werden, wenn sich am Inhalt von TextFeld etwas ändert. Dies wird im TextChange-Event-Handler von TextFeld erledigt.

Ein Event-Handler ist eine Methode, die automatisch abläuft, wenn ein bestimmtes Ereignis eintritt. Jedes Interface-Objekt in REALbasic besitzt einen Satz leerer Event-Handler für Events, die REALbasic automatisch erkennen kann. Programmcode, den Sie in einen dieser Event-Handler eingeben, wird also ausgeführt, wenn das jeweilige Event eintritt. Dieses Konzept nennt man *event-gesteuerte Programmierung*.

Welche Events bei einem EditField verfügbar sind, erfahren Sie, wenn Sie den Code-Editor von TextFenster nach oben bringen und im Browser den Eintrag **Steuerelemente** und dann das TextFeld-Objekt expandieren. Dann sehen Sie eine Liste der Event-Handler eines EditFields:



Um die TextWurdeGeändert-Eigenschaft zu verwalten, gehen Sie folgendermaßen vor:

1. Selektieren Sie den **TextChange**-Event.
2. Fügen Sie folgende Code-Zeile in den leeren Event-Handler ein:
   ```
   TextWurdeGeändert=True
   ```
   Da Sie diese Code-Zeile im TextChange-Event-Handler abgelegt haben, wird sie immer dann automatisch ausgeführt, wenn Änderungen am Text im TextFeld vorgenommen werden. REALbasic übernimmt den Job herauszufinden, ob es Änderungen gibt.

   In der Online-Referenz werden die für jedes Steuerelement verfügbaren Event-Handler beschrieben.

Behandeln des Menüeintrags

Der Menu-Handler für den Menüpunkt **Sichern** ruft die DateiSichern-Methode auf, die Sie gerade implementiert haben und übergibt den Wert **False** an **SichernDialogZeigen**, um zu verhindern, dass die Dateiauswahlbox angezeigt wird (außer, es handelt sich um ein bisher ungesichertes Dokument).

1. Wählen Sie den Menüpunkt **Bearbeiten/Neuer Menu-Handler**. Wählen Sie **AblageSichern** aus dem Popup-Menü aus und klicken Sie auf **OK**.

   Ein neuer Menu-Handler namens **AblageSichern** erscheint im Browser-Bereich des Code-Editors.
2. Geben Sie folgenden Code ein:
   ```
   DateiSichern Title, False
   ```
   Dieser Menu-Handler ruft die Methode DateiSichern auf, die die eigentliche Arbeit erledigt. Die dem Methodennamen folgenden Begriffe – Title und False – sind die Parameter, die an die DateiSichern-Methode übergeben werden. Machen Sie sich noch einmal klar, dass DateiSichern zwei Parameter erwartet. Beim ersten (einer Zeichenkette) handelt es sich um den Namensvorschlag für die zu sichernde Datei. Der zweite Parameter ist vom Typ Boolean und legt fest, ob eine „Änderungen sichern"-Dialogbox angezeigt werden soll (die Sie noch programmieren müssen) oder nicht. Bei dem Ausdruck „Title" handelt es sich um die Title-Eigenschaft der Window-Klasse, die die Zeichenkette enthält, die im Titelbalken als Fenstertitel angezeigt wird.

   Wenn Sie ein Dokument zum ersten Mal speichern, wird „Namenlos" als Namensvorschlag verwendet, da dies der Standard-Fenstertitel ist.
3. Sichern Sie Ihr Projekt.
4. Wählen Sie **Debug/Programm starten**, um Ihre Applikation zu testen. Beachten Sie, dass der Menüpunkt **Sichern** solange inaktiv (disabled) bleibt, bis Sie den Text im Texteditor ändern.

5. Beenden Sie Ihr Programm, um zur Entwicklungsumgebung zurückzukehren.

Hinzufügen eines Menüpunkts Sichern unter

Der Menüpunkt **Sichern unter** erfüllt die gleiche Funktion wie der Menüeintrag **Sichern**, mit dem Unterschied, dass er immer die Dateiauswahlbox anzeigt, die es dem Anwender ermöglicht, das Dokument unter einem neuen Namen abzuspeichern. Er wird auf die gleiche Weise wie der Sichern-Menüpunkt angelegt.

Gehen Sie dazu folgendermaßen vor:

1. Führen Sie einen Doppelklick auf den **Menu**-Eintrag im Projekt-Fenster aus und fügen Sie dem **Ablage**-Menü (Windows: **Datei**-Menü) einen neuen Menüpunkt mit dem Text **Sichern unter…** hinzu. Ändern Sie die Name-Eigenschaft – falls erforderlich – auf **AblageSichernunter**.
2. Draggen Sie den neuen Menüpunkt zwischen die Einträge **Sichern** und **Beenden**.
3. Öffnen Sie den **EnableMenuItems-Event** im Code-Editor von TextFenster und hängen Sie folgende Zeile ans Ende der Methode an:
   ```
   AblageSichernunter.Enable
   ```
4. Rufen Sie bei aktivem Code-Editor von TextFenster den Menüpunkt **Bearbeiten/Neuer Menu-Handler** auf und wählen Sie dort **AblageSichernunter** aus.
5. Geben Sie folgende Code-Zeile ein:
   ```
   DateiSichern Title, True
   ```
   Dieser Menu-Handler erledigt das Sichern ebenfalls mit Hilfe der DateiSichern-Methode, erzwingt jedoch die Anzeige der Dateiauswahlbox.
6. Speichern Sie Ihr Projekt und testen Sie die **Sichern**- und **Sichern unter**- Befehle durch Anwählen von **Debug/Programm starten**. Geben Sie ein wenig Text ein und speichern Sie ihn mit dem **Sichern**-Befehl. Verwenden Sie dann den **Sichern unter**-Befehl. Beachten Sie, dass der **Sichern unter**-Befehl den Titel des TextFensters als Namensvorschlag anbietet.
7. Beenden Sie Ihr Programm, um zur Entwicklungsumgebung zurückzukehren.

Hinzufügen des Öffnen-Menüeintrags

Nachdem Sie nun die Routinen für „Sichern“ und „Sichern unter“ implementiert haben, benötigt der Anwender noch eine Möglichkeit, die Dokumente, die er gespeichert hat, wieder zu laden. Die folgende Übung implementiert einen **Öffnen**-Menüeintrag. Sie werden:

- einen **Öffnen**-Menüeintrag anlegen,
- den Menüeintrag aktivieren,
- einen Menu-Handler für den Eintrag schreiben.

Erzeugen des Öffnen-Menüeintrags

Um den **Öffnen**-Menüeintrag zu erzeugen, gehen Sie folgendermaßen vor:

1. Führen Sie einen Doppelklick auf dem **Menu**-Eintrag im Projekt-Fenster aus und selektieren Sie den leeren Eintrag im **Ablage**-Menü (Windows: **Datei**-Menü). Falls der Menü-Editor sich nicht öffnen lässt, schauen Sie nach, ob der Menüpunkt **Debug/Kill** aktiv ist. Wenn ja, läuft das Programm noch in der Runtime-Umgebung und muss erst beendet oder mit **Debug/Kill** abgebrochen werden.
2. Geben Sie im Eigenschaften-Fenster für den neuen Menüpunkt als **Text**-Eigenschaft **Öffnen...** ein und **O** als **CommandKey**-Eigenschaft. Die Name-Eigenschaft ändern Sie in **AblageOeffnen** (mit „Oe“ statt „Ö“).
3. Draggen Sie den **Öffnen**-Menüpunkt zwischen den **Neu**- und den **Sichern**-Menüeintrag.

Verarbeiten des Menüeintrags

Da das Öffnen-Menü auch dann funktionieren muss, wenn keine Fenster offen sind, gehört sein Menü-Handler in das Application-Objekt.

Um den Öffnen-Menüeintrag zu verwalten, ist folgendes zu tun:

1. Rufen Sie bei aktivem Code-Editor der App-Klasse **Bearbeiten/Neuer Menu-Handler** auf, um einen Menu-Handler für den **Öffnen**-Menüpunkt zu erzeugen.
2. Selektieren Sie im Popup den **Ablageoeffnen**-Menu-Handler, klicken Sie **OK** und geben Sie folgenden Code in den Code-Editor ein:

```
Dim f as FolderItem
Dim w as TextFenster
f=GetOpenFolderItem("text") //Datei öffnen-Dialog anzeigen
If f<>nil then //Der Anwender hat auf Öffnen geklickt
    w=New TextFenster //Neue Instanz von TextFenster erzeugen
    f.OpenStyledEditField w.TextFeld
    w.Dokument=f //weise f der Dokument-Eigenschaft von TextFenster zu
    w.Title=f.name //weise Namen von f dem Titel von TextFenster zu
End if
```

   Die ersten zwei Zeilen erzeugen ein neues FolderItem-Objekt (eine Referenz auf ein Dokument) und eine neue Instanz der TextFenster-Klasse zur Anzeige des Dokuments. An dieser Stelle hat **f** noch keinen Wert, sondern ist lediglich ein Container, der in der Lage ist, auf ein Dokument zu verweisen. Genauso verweist **w** nicht auf eine neue Instanz von TextFenster, solange diese nicht mit der Funktion New erzeugt wurde.

   Als nächstes wird die globale Methode GetOpenFolderItem aufgerufen, die den „Datei öffnen"-Dialog anzeigt und eine Referenz auf das vom Anwender ausgewählte Dokument zurückliefert. Der Parameter „text" teilt der GetOpenFolderItem-Methode mit, dass nur Textdateien in der Dateiliste erscheinen sollen. Den Dateityp „text" haben Sie dem Projekt ja bereits hinzugefügt.

   Wenn der Anwender erfolgreich ein Text-Dokument öffnet, liefert die GetOpenFolderItem-Funktion eine Referenz auf das Dokument im FolderItem-Objekt **f**.

   Der Code testet zuerst, ob **f** immer noch den Wert Nil hat – das wäre der Fall, wenn der Anwender den „Datei öffnen"-Dialog mit dem Abbrechen-Knopf verlassen hat. Ist **f** ungleich Nil, wird ein neues Dokumentfenster geöffnet und die OpenStyledEditField-Methode der FolderItem-Klasse aufgerufen. Diese Methode platziert den Text, der nun **f** ist, in das TextFeld, das zu der neu erzeugten Instanz von TextFenster gehört.

   Schließlich wird der Dokument-Eigenschaft des TextFensters das FolderItem (also das vom Anwender geöffnete Dokument) zugewiesen und als Fenstertitel der Name des Dokuments eingesetzt.

   Informationen zu FolderItem, GetFolderItem und Window-Klasse finden Sie in der Online-Referenz.
3. Speichern Sie Ihr Projekt.
4. Wechseln Sie in die Runtime-Umgebung und testen Sie die Öffnen- und Sichern-Kommandos.

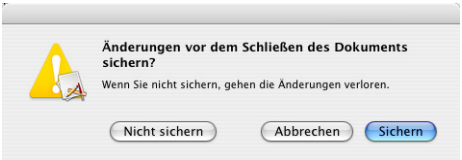
   Wenn Sie Fehlermeldungen erhalten wie „Unbekannter Bezeichner" oder andere Probleme auftreten, überprüfen Sie, ob Sie alle Objekte umbenannt haben und den Code jeweils in den richtigen Code-Editor eingegeben haben. Wenn Sie trotzdem weiterhin Probleme haben, laden Sie das „TextEditor-Kapitel4"-Projekt von der CD und vergleichen es mit Ihrem Projekt.
5. Beenden Sie die Applikation und kehren Sie zur Entwicklungsumgebung zurück.

Rückblick

In diesem Kapitel haben Sie Ihr Programm um die Fähigkeiten Dokumente zu laden, zu erzeugen, zu schließen und zu speichern erweitert.

5. Hinzufügen einer „Änderungen sichern“-Dialogbox

Ein benutzerfreundliches Programm gibt dem Anwender die Gelegenheit, Änderungen in offenen Dokumenten zu sichern, wenn er ein Fenster schließt oder das Programm beenden will. Unser Programm macht da keine Ausnahme. In dieser Übung werden Sie die unten abgebildete **Änderungen sichern**-Dialogbox erzeugen.



Erzeugen der Dialogbox

Zum Erzeugen der Dialogbox werden wir eine in REALbasic integrierte Klasse – die MessageDialog-Klasse verwenden. Diese dient zum Anzeigen kurzer Mitteilungen und bietet dem Anwender auch eine Interaktionsmöglichkeit. Wenn Sie MessageDialog zum Anzeigen einer Dialogbox verwenden, müssen Sie dafür nicht extra ein Fenster erzeugen und zu Ihrer Projektdatei hinzufügen.

Die MessageDialog-Klasse verwendet nicht den Fenster-Editor. Statt dessen erzeugen Sie in Ihrem Code eine Instanz der MessageDialog-Klasse und verwenden ihre Eigenschaften, um das Aussehen des Dialogs zu bestimmen.

Ein MessageDialog-Fenster kann folgende Elemente enthalten:

Eigenschaft	Beschreibung
Icon	Symbol, das links neben der Nachricht angezeigt wird.
Message	Nachricht, die im Dialog angezeigt wird.
Explanation	Zusätzliche Erklärung, die kleiner als die eigentliche Nachricht angezeigt wird.
ActionButton	Löst die Standardaktion aus.
CancelButton	Bricht die Aktion ab.
AlternateActionButton	Löst eine alternative Aktion aus.
Title	Überschrift für die Titelzeile des Dialogs. Wird auf dem Mac nicht angezeigt.

Die MessageDialog-Klasse sorgt automatisch für die richtige Größe des Dialogs und die Platzierung der Elemente.

Hinweis: Die Windows-Version der MessageDialog-Klasse ordnet die Knöpfe in einer anderen Reihenfolge als die Macintosh-Version an und unterstützt zusätzlich die Title-Eigenschaft.

Jeder Knopf eines MessageDialogs ist ein MessageDialogButton-Objekt. Ein MessageDialogButton hat spezielle Eigenschaften, die in der folgenden Tabelle aufgeführt sind.

Eigenschaft	Beschreibung
Caption	Die Beschriftung des Knopfes.
Visible	Auf True setzen, um den Knopf anzuzeigen.
Default	Auf True setzen, um den Knopf als Standardknopf zu hervorzuheben. Standardmäßig ist diese Eigenschaft für die Klasse ActionButton auf True gesetzt.
Cancel	Auf True setzen, damit dieser Knopf auf das Drücken der Esc-Taste reagiert. Unter Mac OS reagiert dieser ebenso auf das Drücken von ⌘-„.“. Diese Eigenschaft kann nur für einen Knopf auf True gesetzt sein. Einige Betriebssysteme erlauben nicht, dass der Cancel-Knopf gleichzeitig Standardknopf ist.

Sie müssen nun eine Methode schreiben, die eine Instanz der MessageDialog-Klasse erzeugt, ihren Eigenschaften die korrekten Werte zuweist und abhängig vom angeklickten Knopf die passenden Aktionen ausführt.

Eigenschaften des MessageDialogs festlegen

Standardmäßig wird nur der ActionButton eines MessageDialogs angezeigt. Um die beiden anderen Knöpfe anzuzeigen, muss deren Visible-Eigenschaft auf True gesetzt werden. Außerdem muss das Symbol festgelegt werden. Der folgende Code legt das Aussehen der „Änderungen sichern"-Dialogbox fest:

```
Dim d as New MessageDialog //MessageDialog-Objekt deklarieren
Dim b as New MessageDialogButton //Zum Verarbeiten des Ergebnisses
d.icon=1 //Warnsymbol zeigen
d.ActionButton.Caption="Sichern"
d.CancelButton.Visible=True //Abbrechen-Knopf zeigen
d.CancelButton.Caption="Abbrechen"
d.AlternateActionButton.Visible=True //Nicht Sichern-Knopf zeigen
d.AlternateActionButton.Caption="Nicht sichern"
d.Message="Änderungen vor dem Schließen des Dokuments sichern?"
d.Explanation="Wenn Sie nicht sichern, gehen die Änderungen verloren."
```

Interessant an diesem Programmcode ist die Zeile `d.ActionButton.Caption="Sichern"`. Dabei wurde die Punkt-Notation, die Sie bereits verwendet haben, um eine Eigenschaft eines Objekts anzusprechen, dahingehend erweitert, dass nun eine Eigenschaft eines Objekts, das zu einem anderen Objekt gehört, angesprochen wird.

Benutzereingaben verarbeiten

Nachdem Sie den MessageDialog „konstruiert" haben, müssen Sie ihn dem Anwender präsentieren und auf dessen Knopfdruck entsprechend reagieren. Darum kümmert sich der folgende Code-Schnipsel. Die Variable b wird dabei als MessageDialogButton deklariert und enthält den Knopf, den der Anwender angeklickt hat.

```
b=d.ShowModal //Dialog anzeigen und auf Benutzereingaben warten
Select Case b //Feststellen, welcher Knopf angeklickt wurde
   Case d.ActionButton //Anwender hat Sichern angeklickt
      DateiSichern Title,False
      Close
   Case d.AlternateActionButton
      //Anwender hat Nicht sichern angeklickt
      //Fenster kann ohne Sichern der Änderungen geschlossen werden
   Case d.CancelButton //Anwender hat Abbrechen angeklickt
   Return True //Schließen des Fensters abbrechen
End select
```

Die ShowModal-Methode der MessageDialog-Klasse liefert einen MessageDialogButton zurück (den Sie im ersten Teil des Quelltextes deklariert hatten. Bei diesem MessageDialogButton handelt es sich um den Knopf, den der Anwender angeklickt hat.

Um welchen der drei Knöpfe es sich dabei handelt, ermittelt die Select-Case-Anweisung, die die drei Möglichkeiten testet. Jede Select-Anweisung der Select-Case-End-Struktur testet das MessageDialogButton-Objekt b auf einen konkreten Wert. Ist b der ActionButton, wird die DateiSichern-Methode ausgeführt und der Inhalt des aktuellen Dokuments gespeichert. Ist b der AlternateActionButton, passiert gar nichts. Ist b der CancelButton, wird das Schließen des Fensters bzw. das Beenden der Applikation abgebrochen und der CancelClose-Event-Handler liefert True zurück.

Integration der „Änderungen sichern"-Dialogbox in Ihr Projekt

Der letzte Schritt besteht darin, den „Änderungen sichern"-Dialog anzuzeigen, wenn der Anwender den Beenden-Menüpunkt aufruft oder ein Dokumentfenster schließt. Dies wird im CancelClose-Event-Handler von TextFenster realisiert.

Der Menüpunkt **Ablage/Beenden** (Windows: **Datei/Beenden**) unterscheidet sich von den Menüpunkten, die Sie selbst angelegt haben. So ist **Beenden** eine Instanz der **QuitMenuItem**-Klasse und nicht der **MenuItem**-Klasse. Sie

können dies überprüfen, indem Sie den Menüpunkt **Beenden** im Menü-Editor selektieren und sich seine Eigenschaften anschauen.

Außerdem ist das **Beenden**-Menü per Vorgabe aktiviert. Sie haben es schon mehrfach benutzt, um nach einem Testlauf unseres Beispielprojekts in die Entwicklungsumgebung zurückzukehren, ohne dass Sie es vorher aktivieren mussten.

Schließlich hat das **Beenden**-Menü auch seinen eigenen Menü-Handler, der die in REALbasic integrierte **Quit**-Methode aufruft. Diese versucht, das Programm zu beenden. Dazu ruft sie für jedes offene Fenster den **CancelClose**-Event-Handler auf. Dieser Event-Handler ermöglicht Ihnen, das Beenden der Applikation zu unterbrechen, um vorher noch bestimmte Aktionen auszuführen (z.B. ungesicherte Änderungen an Dokumenten abzuspeichern).

Falls der CancelClose-Event **False** zurückliefert (das ist normalerweise der Fall und besagt, dass das Beenden des Programms nicht abgebrochen werden soll), wird der **Close**-Event-Handler des Fensters ausgeführt. Wenn der **CancelClose**-Event **True** zurückgibt, schickt REALbasic keine weiteren **CancelClose**- oder **Close**-Events und das Programm wird nicht beendet.

Die **CancelClose**-Methode, die Sie jetzt implementieren werden, öffnet die **Änderungen sichern**-Dialogbox, falls die **TextWurdeGeändert**-Eigenschaft den Wert **True** hat. Sie ermittelt dann, welchen Knopf der Benutzer in der **Änderungen sichern**-Dialogbox gedrückt hat. Nur wenn der Benutzer auf den **Sichern**-Knopf geklickt hat, wird die **DateiSichern**-Methode aufgerufen.

Um den **CancelClose**-Code hinzuzufügen, ist folgendes zu tun:

1. Expandieren Sie im Code-Editor von **TextFenster** den **Events**-Eintrag und selektieren Sie den **CancelClose**-Eintrag. *Achtung:* Sie befinden sich nun wieder im Code-Editor für TextFenster.
2. Geben Sie folgenden Code ein:

```
Dim d as New MessageDialog //MessageDialog-Objekt deklarieren
Dim b as New MessageDialogButton //Zum Verarbeiten des Ergebnisses
If TextHasChanged then
    d.icon=1 //Warnsymbol zeigen
    d.ActionButton.Caption="Sichern"
    d.CancelButton.Visible=True //Abbrechen-Knopf zeigen
    d.CancelButton.Caption="Abbrechen"
    d.AlternateActionButton.Visible=True //Nicht Sichern-Knopf zeigen
    d.AlternateActionButton.Caption="Nicht sichern"
    d.Message="Änderungen vor dem Schließen des Dokuments sichern?"
    d.Explanation="Wenn Sie nicht sichern, gehen die Änderungen verloren."
    b=d.ShowModal //Dialog anzeigen und auf Benutzereingaben warten
    Select Case b //Feststellen, welcher Knopf angeklickt wurde
        Case d.ActionButton //Anwender hat Sichern angeklickt
            DateiSichern Title,False
            Close
    Case d.AlternateActionButton
           //Anwender hat Nicht sichern angeklickt
           //Fenster kann ohne Sichern der Änderungen geschlossen werden
    Case d.CancelButton //Anwender hat Abbrechen angeklickt
        Return True //Schließen des Fensters abbrechen
    End select
End if
```

Dieser Code testet, ob der Text verändert wurde, indem er den Wert der TextWurdeGeändert-Eigenschaft abfragt. Ist dieser True, wird die Änderungen sichern-Dialogbox angezeigt.

Die Anweisung d.ShowModal führt die ShowModal-Methode der MessageDialog-Klasse aus und liefert den MessageDialogButton zurück, den der Anwender angeklickt hat.

Die Select-Case-Struktur stellt fest, welchen Knopf der Anwender gedrückt hat. Falls er Nicht sichern angeklickt hat, wird das Beenden des Programms fortgesetzt, da CancelClose in diesem Fall False zurückliefert und keine Methode zum Speichern des Dokuments aufgerufen wird. Wenn der Anwender Abbrechen anklickt, liefert CancelClose den Wert True zurück und das Programm wird nicht beendet. Klickt der Anwender auf Sichern, so wird die

DateiSichern-Methode ausgeführt. Diese erhält als Parameter den Namen des Fensters (die Title-Eigenschaft der Window-Klasse enthält den Namen des jeweiligen Fensters) und False (dies teilt der DateiSichern-Methode mit, dass sie nicht die Dateiauswahlbox anzeigen soll).

3. Speichern Sie Ihr Projekt unter **TextEditor-Kapitel5**. Der Code-Editor für die CancelClose-Methode sollte nun so aussehen:

4. Testen Sie die „Änderungen sichern"-Dialogbox, indem Sie in die Runtime-Umgebung wechseln, ein neues Dokument erzeugen, es speichern und es dann modifizieren. Wenn Sie danach Ihr Programm beenden, sollte die „Änderungen sichern"-Dialogbox erscheinen. Wenn Sie auf „Nicht sichern" klicken, sollte sich Ihr Programm beenden. Wenn Sie auf „Sichern" klicken, erscheint eine Dateiauswahlbox. Wenn Sie auf „Abbrechen" klicken, wird der Vorgang abgebrochen und das Programm nicht beendet.

Rückblick

In diesem Kapitel haben Sie gelernt, wie eine MessageDialog-Box funktioniert.

6. Drag und Drop einbauen

In diesem Kapitel erweitern Sie das Programm um die Fähigkeit, Text-Dokumente einfach durch Draggen vom Desktop auf ein offenes Fenster zu öffnen. Sie können mehrere Dokumente auf einmal draggen. Dabei kann das Zielfenster entweder leer sein oder schon ein Dokument enthalten.

Sie werden lernen, wie man

- ein EditField-Steuerelement so konfiguriert, dass es gedraggte Dateien akzeptiert,
- mehrere gedraggte Dateien abarbeitet.

EditFields unterstützen automatisch Drag & Drop innerhalb von REALbasic, wenn die MultiLine-Eigenschaft gesetzt ist. Die Schritte in diesem Kapitel sind nötig, um externe Dokumente per Drag & Drop zu verarbeiten.

Los geht's

Wenn Ihr Projekt nicht bereits geöffnet ist, laden Sie die Datei „**TextEditor-Kapitel5**" von Ihrer REALbasic-CD.

TextFeld für Drag & Drop von Dokumenten vorbereiten

Der erste Schritt besteht darin, TextFeld so zu konfigurieren, dass es gedraggte Textdateien akzeptiert.

Bevor ein Steuerelement gedraggte Objekte akzeptiert, müssen Sie festlegen, welche Dateitypen es akzeptieren soll. Dies geschieht im **Dateitypen**-Dialog.

Da **TextFeld** immer gedraggte Dateien akzeptieren soll, sorgen Sie im Open-Event-Handler von **TextFeld** dafür. Dieses Event läuft als erstes ab, wenn ein Dokument-Fenster geöffnet wird.

Um den Textdateien-Typ hinzuzufügen, ist folgendes zu tun:

1. Expandieren Sie im Code-Editor von TextFenster unter Steuerelemente das **TextFeld**-Objekt und öffnen Sie den **Open**-Event-Handler.
2. Fügen Sie dieser Methode folgenden Code hinzu:
   ```
   // erlaubt Drag & Drop von Textdateien
   Me.AcceptFileDrop("text")
   ```
   Mit der Methode **AcceptFileDrop** legen Sie fest, welche Dateitypen auf ein Steuerelement gedraggt werden dürfen. Der Parameter „text" ist der Name des Dateityps, den Sie in Kapitel 4 definiert haben.

Schließlich müssen Sie festlegen, wie sich **TextFeld** verhalten soll, wenn der Anwender eine oder mehrere Dateien dieses Typs gedraggt hat. Dazu bietet sich der **DropObject**-Event-Handler an.

Um gedraggte Dateien abzuarbeiten, ist folgendes zu tun:

1. Selektieren Sie den **DropObject**-Event-Handler von TextFeld
   Dieser bekommt den Parameter „**obj As DragItem**" übergeben. Über die Eigenschaften der **DragItem**-Klasse können Sie feststellen, welche Datentypen auf das Objekt gedraggt wurden. Wenn Dateitypen gedraggt wurden, die akzeptiert werden, können Sie über andere Eigenschaften die Daten auslesen.
2. Fügen Sie folgenden Code in den DropObject-Event-Handler ein:
   ```
   Dim TextStream as TextInputStream
   If obj.FolderItemAvailable then
      Do
         TextStream=obj.FolderItem.OpenAsTextFile
         me.text=me.text+TextStream.ReadAll+Chr(13)
      loop until not obj.nextItem
   End if
   ```
3. Speichern Sie das Projekt unter **TextEditor-Kapitel6**.

Die Eigenschaft **FolderItemAvailable** des **DragItem**-Objekts ist **True**, wenn ein oder mehrere **FolderItems** (das sind die Textdokumente) gedraggt wurden.

Wenn ein **FolderItem** verfügbar ist, verwendet der Code die **OpenAsTextFile**-Methode der **FolderItem**-Klasse, um die Textdatei zu öffnen. Diese Methode liefert ein Objekt vom Typ **TextInputStream**, das den Inhalt der Textdatei enthält. Die **ReadAll**-Methode wird verwendet, um den Inhalt des **FolderItems** an die **Text**-Eigenschaft von **TextFeld** anzuhängen. Die **Chr**-Funktion fügt ein Return-Zeichen an das Ende des Textes an.

Da der Anwender mehrere Dateien gleichzeitig draggen kann, wird die **Do**...**Loop**-Schleife verwendet, um diesen Prozess so lange zu wiederholen, bis keine weiteren akzeptierten **FolderItems** mehr verfügbar sind. Die **NextItem**-Methode weist den Eigenschaften des **DragItem**-Objekts die Werte des nächsten verfügbaren Objekts zu und liefert **False**, wenn keine weiteren passenden Objekte verbleiben.

Testen der Applikation

Nachdem Sie nun Drag & Drop eingebaut haben, können Sie die Applikation testen.

1. Wählen Sie **Debug/Programm starten** und experimentieren Sie mit verschiedenen Text-Dateien und Text-Clips.
   Sie werden bemerken, dass **TextEditor** gedraggte Objekte mit falschem Dateityp zurückweist.
   Sie können auch Text im **TextEditor** selektieren und auf den Schreibtisch draggen, um einen Text-Clip zu erzeugen.
2. Beenden Sie Ihr Programm, um in die Entwicklungsumgebung zurückzukehren.

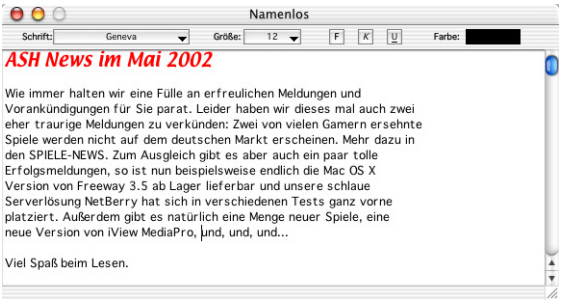
Rückblick

In diesem Kapitel haben Sie den TextEditor um Drag & Drop-Fähigkeiten erweitert.

7. Arbeiten mit Text

In diesem Kapitel werden Sie mit dem Styled-Text-Feature (Text mit Stilinformationen) von REALbasic arbeiten. Sie werden ein Stil-, Größe- und Farbe-Menü implementieren. Diese Menüs erlauben es dem Benutzer, einem selektierten Text einen bestimmten Schriftstil und eine bestimmte Schriftgröße zuzuweisen. Außerdem erfahren Sie, wie man im Menü ein Häkchen neben den gerade gewählten Schriftstil und die gewählte Schriftgröße setzt, damit der Benutzer die aktuellen Einstellungen ablesen kann.

Im Kapitel 8 werden Sie ein Font-Menü hinzufügen, das dem Anwender gestattet, jeden auf dem Rechner installierten Zeichensatz zu verwenden. Wenn Sie damit fertig sind, wird Ihr Dokument-Fenster folgendermaßen aussehen:



Die Steuerelemente für die Schrift und die Schriftgröße sind Popup-Menüs, die Stil-Steuerelemente sind Knöpfe und das Farbe-Steuerelement ist ein Canvas-Control, das die Farbe des selektierten Textes anzeigt und die Farbauswahl aufruft, wenn der Anwender darauf klickt.

Los geht's

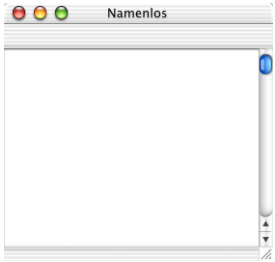
Laden Sie Ihr REALbasic-Projekt **TextEditor-Kapitel6**.

Konfigurieren von TextFeld für Styled Text

Bevor Sie die Stil- und Größe-Menüs implementieren, müssen Sie dem TextFeld beibringen, Text mit verschiedenen Stilen und Größen zu verarbeiten. Dazu müssen Sie für TextFeld die **Styled**-Eigenschaft setzen. Außerdem müssen Sie den oberen Rand von TextFeld ein paar Pixel nach unten verschieben, damit Platz für die Steuerelemente entsteht.

1. Führen Sie einen Doppelklick auf den **TextFenster**-Eintrag im Projektfenster aus.
   Das Fenster öffnet sich im Fenster-Editor. Seine Eigenschaften werden im **Eigenschaften**-Fenster angezeigt.
2. Klicken Sie auf das **TextFeld** im Fenster (also auf den weißen Fensterhintergrund) und aktivieren Sie im Eigenschaften-Fenster die **Styled**-Eigenschaft. Die Styled-Eigenschaft finden Sie im Abschnitt **Aussehen**.
   Ohne aktivierte Styled-Eigenschaft wären Sie nicht in der Lage, verschiedenen Textabschnitten verschiedene Stile zuzuweisen.
3. Setzen Sie die **Top**-Eigenschaft von TextFeld auf 25.

4. Reduzieren Sie den Wert der **Height**-Eigenschaft um 26.
   Ihr TextFenster sollte nun etwa so aussehen:



5. Speichern Sie das Projekt unter **TextEditor-Kapitel7**.

Anlegen eines Fontgröße-Popup-Menüs

In diesem Abschnitt erzeugen Sie ein Größe-Menü und seine Menüeinträge.

Das Größe-Menü und seine Menüeinträge erzeugen

1. Ziehen Sie ein StaticText-Steuerelement Aa aus der Steuerelemente-Palette in den oberen Bereich von **TextFenster**.
2. Stellen Sie die Eigenschaften dieses StaticText-Steuerelements wie folgt ein:

Eigenschaft	Wert
Left	203
Top	4
Width	43
Height	16
Text	Größe:
TextAlign	Rechts
TextFont	System
TextSize	0
Bold	True

   Sie fragen sich vielleicht, weshalb Sie die **TextSize**-Eigenschaft auf Null setzen sollen. Diese Eigenschaft legt die Größe des sichtbaren Texts im **StaticText** fest. Sie können **TextSize** entweder auf eine bestimmte Schriftgröße oder auf Null setzen, um REALbasic mitzuteilen, dass es die Standardgröße für die Plattform verwenden soll, auf der Ihre Applikation läuft. Die optimale Schriftgröße für eine Applikation ist unter Mac OS, Windows und Linux normalerweise unterschiedlich. Wenn Ihre Applikation nur auf einer Plattform zum Einsatz kommen soll, können Sie natürlich eine konkrete Schriftgröße vorgeben.

3. Draggen Sie als nächstes ein **BevelButton**-Steuerelement ad rechts neben das **StaticText**-Steuerelement.
   Ein BevelButton kann entweder als PushButton oder als Popup-Menü konfiguriert werden und entweder Text oder ein Bild darstellen.

4. Stellen Sie mit dem Eigenschaften-Fenster folgende Werte für den BevelButton ein:

Eigenschaft	Wert
Name	GroesseMenue
Left	251
Top	4
Width	43
Height	16
Caption	(leer lassen)
CaptionAlign	Zentriert
CaptionPlacement	Normal
HasMenu	Normales Menü
TextFont	System
TextSize	0

   Die Eigenschaft HasMenu führt dazu, dass sich der BevelButton wie ein Popup-Menü und nicht wie ein Knopf verhält. Als nächstes müssen Sie die Menüeinträge erzeugen. Das soll zur Laufzeit des Programms genau dann geschehen, wenn das Fenster geöffnet wird.

5. Öffnen Sie im Code-Editor von TextFenster unter den Steuerelementen den Eintrag **GroesseMenue** und aktivieren Sie den **Open**-Event-Handler.

6. Geben Sie folgenden Code in den Event-Handler ein:

```
me.addrow "9"
me.addrow "10"
me.addrow "12"
me.addrow "14"
me.addrow "18"
me.addrow "24"
me.addrow "36"
me.caption=Str(TextFeld.SelTextSize)
```

Die ersten sieben Zeilen rufen jeweils die **AddRow**-Methode der **BevelButton**-Klasse auf, um in das Popup-Menü einen neuen Eintrag einzufügen. Die letzte Zeile setzt den Vorgabewert der **Caption**-Eigenschaft auf die Schriftgröße an der Cursorposition. Dabei konvertiert die **Str**-Funktion die Schriftgröße des selektierten Textes im TextFeld (also einen Integer-Wert)in einen String. Sie müssen diese Konvertierung vornehmen, da die Caption-Eigenschaft nur Strings akzeptiert. Wenn Sie versuchen, einen numerischen Wert zu übergeben, erhalten Sie eine Fehlermeldung.

Der Ausdruck **Me** verweist auf das Steuerelement, zu dem der Event-Handler gehört, also **GroesseMenue**. Sie hätten auch **GroesseMenue.addrow** schreiben können, wenn Sie jedoch **Me** verwenden, ist der Code universeller und kann in andere BevelButton-Steuerelemente kopiert werden, ohne dass er angepasst werden muss. Er funktioniert außerdem auch dann noch, wenn Sie den Namen des Steuerelements ändern.

Jetzt legen Sie noch fest, was passieren soll, wenn der Anwender einen Menüeintrag aufruft. Dies passiert im **Action**-Event-Handler von **GroesseMenue**. Er wird ausgeführt, wenn der Anwender in dem Menü eine Auswahl trifft.

7. Geben Sie folgenden Code in den **Action**-Event-Handler ein:

```
Me.Caption=Me.List(Me.MenuValue)
TextFeld.SelTextSize=Val(Me.Caption)
```

Die Eigenschaft **MenuValue** ist die *Nummer* des gewählten Menüeintrags und die **List**-Methode liefert den Text des Menüeintrags, der zur übergebenen Nummer gehört. Die erste Zeile setzt die **Caption**-Eigenschaft (der Text, der im Steuerelement angezeigt wird) auf die Schriftgröße, die der Anwender ausgewählt hat.

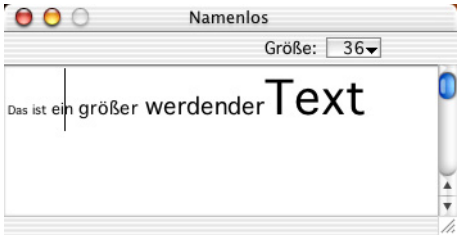
Die zweite Zeile weist der **SelTextSize**-Eigenschaft von TextFeld die gewählte Schriftgröße zu. Damit wird die Größe von eventuell im Dokumentfenster selektiertem Text entsprechend geändert. Außerdem erscheint neu eingegebener Text in dieser Größe.

Austesten des Größe-Menüs

Speichern Sie Ihr Projekt und testen Sie es mit **Debug/Programm starten**. Tippen Sie ein paar Wörter und versuchen Sie, die Schriftgröße zu ändern.

Ein ungelöstes Problem

Wenn Sie verschiedenen Wörtern verschiedene Schriftgrößen zuweisen und den Cursor durch den Text bewegen, werden Sie bemerken, dass das Größe-Menü nicht die aktuelle Schriftgröße anzeigt. Das bedeutet, dass das Größe-Menü mit dem TextFeld kommunizieren kann, aber keinerlei Rückmeldung von TextFeld über die Schriftgröße des selektierten Textes bekommt. Dieses Problem zeigt folgende Abbildung:



Das Größe-Menü zeigt **36** an, da das Wort „Text“ zuvor in Größe 36 geschrieben wurde. Wenn dann der Cursor auf das in Größe 12 geschriebene „ein“ gesetzt wird, zeigt das Größe-Menü immer noch 36. Das ist natürlich unschön.

Das Fontgröße-Menü aktualisieren

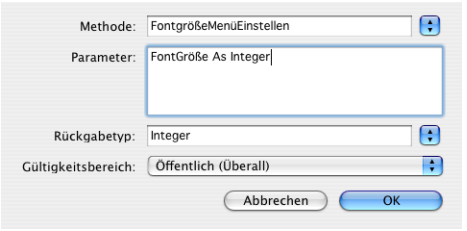
Um das Schriftgröße-Menü zu aktualisieren, verwenden Sie den **SelChange**-Event von TextFeld. Er wird immer dann ausgeführt, wenn der Anwender die Textselektion ändert. Das schließt das Bewegen des Cursors ein.

1. Öffnen Sie den Code-Editor von TextFenster, öffnen Sie das **TextFeld**-Steuerelement und wählen Sie den **SelChange**-Event-Handler aus.
2. Geben Sie folgenden Code in diesen Event-Handler ein:

```
//Fontgrößemenü aktualisieren
If Str(Me.SelTextSize) <> GroesseMenue.Caption then
   GroesseMenue.Caption=Str(Me.SelTextSize)
   GroesseMenue.MenuValue=FontgrößeMenüEinstellen(Me.SelTextSize)
End if
```

   Der **If**-Befehl überprüft, ob die Schriftgröße des selektierten Textes mit den aktuellen Einstellungen im Menü übereinstimmt. Wenn das nicht der Fall ist, werden die **Caption**- und **MenuValue**-Eigenschaften von **GroesseMenue** entsprechend gesetzt. Um letzteres zu bewerkstelligen, wird eine Funktion benötigt, die die Schriftgröße des selektierten Textes in die Nummer des entsprechenden Menüeintrags konvertiert (die **MenuValue**-Eigenschaft reicht von null bis 6 und enthält nicht den Text des Menüeintrags). Die Methode **FontgrößeMenüEinstellen** ist eine einfache Funktion, die die Konvertierung vornimmt. Sie müssen diese nun dem Projekt hinzufügen.
3. Rufen Sie für den Code-Editor von TextFenster den Menüpunkt **Bearbeiten/Neue Methode** auf. Die Dialogbox **Neue Methode** erscheint.

4. Geben Sie den Namen **FontgrößeMenüEinstellen**, den Parameter **FontGröße As Integer** und den Rückgabetyp **Integer** ein. Die Dialogbox sollte so aussehen:



   Wenn Sie auf OK klicken, öffnet der Code-Editor von TextFenster die **FontgrößeMenüEinstellen**-Methode.

5. Geben Sie folgenden Code in den Code-Editor ein:

```
Dim s as Integer
Select case FontGröße
case 9
    s=0
case 10
    s=1
case 12
    s=2
case 14
    s=3
case 18
    s=4
case 24
    s=5
case 36
    s=6
end select
Return s
```

Der **Select Case**-Befehl ermittelt aus dem der Methode übergebenen Parameter (der Fontgröße) die Nummer des entsprechenden Menüeintrags, weist diese der Variablen s zu und liefert diese zurück.

6. Testen Sie das Programm mit **Debug/Programm starten**. Wenn Sie den Cursor durch einen Text mit unterschiedlichen Schriftgrößen bewegen, aktualisiert sich das Größe-Menü automatisch.

Implementieren des Stil-Menüs

Als nächstes müssen Sie die drei Knöpfe rechts vom Größe-Menü hinzufügen, damit der Anwender die Stile **Fett**, **Kursiv** und **Unterstrichen** zuweisen kann. Zusätzlich könnten Sie auch noch die Stile **Konturschrift**, **Schattiert**, **Eng** und **Breit** verwenden, diese werden allerdings nur unter Mac OS und nicht unter Windows unterstützt. In diesem Tutorial implementieren Sie nur die drei Basis-Stile.

Sie verwenden für den Font-Stil ebenfalls BevelButtons. Diesmal fungieren sie jedoch als Knöpfe und nicht als Popup-Menüs.

Die Stil-Buttons erzeugen

1. Vergrößern Sie TextFenster, indem Sie den Vergrößerungsknopf nach rechts ziehen, um Platz für die zusätzlichen Steuerelemente zu schaffen.

2. Draggen Sie ein BevelButton-Steuerelement  aus der Steuerelemente-Palette in den Kopfbereich von TextFenster, rechts neben das Größe-Menü.

3. Stellen Sie seine Eigenschaften im Eigenschaften-Fenster auf folgende Werte:

Eigenschaft	Wert
Name	KnopfFett
Left	340
Top	4
Width	16
Height	16
Caption	F
CaptionAlign	Zentriert
TextFont	System
TextSize	9
Bold	True
Italic	False
Underline	False
ButtonType	Umschalter

4. Duplizieren Sie den Knopf zweimal (Befehlstaste-D bzw. Strg-D), bewegen Sie die neuen Knöpfe auf ihr ungefähren Positionen rechts neben den **Fett**-Knopf und stellen Sie deren Eigenschaften folgendermaßen ein:

Eigenschaft	Kursiv-Knopf	Unterstrichen-Knopf
Name	KnopfKursiv	KnopfUnterstrichen
Left	370	400
Top	4	4
Width	16	16
Height	16	16
Caption	K	U
CaptionAlign	Zentriert	Zentriert
HasMenu	Kein Menü	Kein Menü
TextFont	System	System
TextSize	9	9
Bold	False	False
Italic	True	False
Underline	False	True
ButtonType	Umschalter	Umschalter

Als nächstes schreiben Sie den Programmcode für die drei Knöpfe. Dies machen Sie im **Action**-Event-Handler eines jeden Knopfes. Dieser wird ausgeführt, wenn der Anwender auf den Knopf klickt.

5. Aktivieren Sie im Code-Editor von TextFenster den **Action**-Event-Handler von **KnopfFett** und fügen Sie folgende Code-Zeile hinzu:

```
TextFeld.ToggleSelectionBold
```

Diese Zeile schaltet das Fett-Attribut von selektiertem Text um. Ist der selektierte Text bereits fett, wird das Fett-Attribut entfernt. Ist der Text noch nicht fett, wird das Fett-Attribut gesetzt.

6. Schreiben Sie folgende Zeile in den Action-Event-Handler von KnopfKursiv:
   `TextFeld.ToggleSelectionItalic`
7. Schreiben Sie folgende Zeile in den Action-Event-Handler von KnopfUnterstrichen:
   `TextFeld.ToggleSelectionUnderline`
8. Speichern Sie Ihr Projekt.

Die Stil-Steuerelemente aktualisieren

Als letzten Schritt müssen Sie den Code hinzufügen, der die Knöpfe Fett, Kursiv und Unterstrichen aktualisiert, wenn der Anwender den Cursor im Text bewegt.

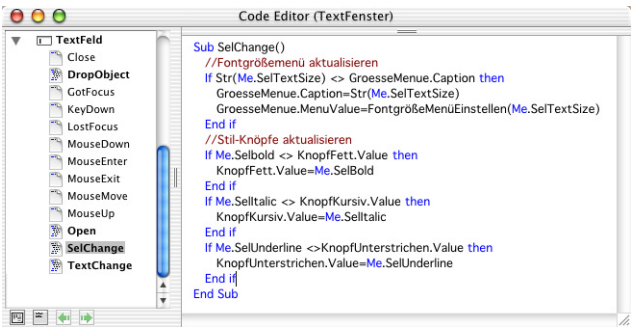
Hierzu müssen Sie zusätzlichen Code in den **SelChange**-Event-Handler von TextFeld eingeben. Dieser Event wird immer ausgeführt, wenn sich die Selektion geändert hat.

1. Expandieren Sie im Code-Editor von TextFenster den Eintrag **Steuerelemente** und expandieren dort den **TextFeld**-Eintrag.
2. Klicken Sie auf **SelChange** und fügen Sie folgenden Code hinter den bereits vorhandenen ein:

```
//Stil-Knöpfe aktualisieren
If Me.Selbold <> KnopfFett.Value then
   KnopfFett.Value=Me.SelBold
End if
If Me.SelItalic <> KnopfKursiv.Value then
   KnopfKursiv.Value=Me.SelItalic
End if
If Me.SelUnderline <>KnopfUnterstrichen.Value then
   KnopfUnterstrichen.Value=Me.SelUnderline
End if
```

Jeder **If**-Befehl überprüft, ob der Zustand eines Knopfes zu dem Stil-Attribut des selektierten Textes passt. Ist dies nicht der Fall, wird der Knopf entsprechend geändert. Wenn zum Beispiel die Value-Eigenschaft von **KnopfFett** True, der selektierte Text aber nicht fett ist, setzt der Code die **Value**-Eigenschaft von **KnopfFett** auf **False**.

Der **SelChange**-Event-Handler sollte nun folgendermaßen aussehen:

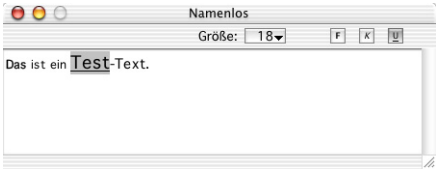


Testen der Stil- und Größe-Steuerelemente

Nachdem nun der Code eingegeben ist, können Sie ausprobieren, ob er funktioniert.

1. Rufen Sie **Debug/Programm starten** auf und geben Sie Text in den Text-Editor ein.
2. Selektieren Sie etwas Text und versuchen Sie, seine Größe und seinen Stil zu ändern.

Wenn ein Stil selektiert ist, ist der zugehörige Knopf ausgewählt. Verwendet der Text keinen Stil, ist keiner der drei Knöpfe ausgewählt. Sollte sich Ihr Programm anders verhalten, haben Sie vermutlich vergessen, die **ButtonType**-Eigenschaft auf **Umschalter** zu stellen.



3. Beenden Sie Ihr Programm, um in die Entwicklungsumgebung zurückzukehren.

Ein Farbe-Steuerelement implementieren

In diesem Abschnitt fügen Sie ein Farbe-Steuerelement hinzu, das dem Anwender gestattet, dem Text ein Farbattribut zuzuweisen. Sie verwenden hierzu ein Canvas-Steuerelement.

Das Canvas-Steuerelement ist eine leere „Leinwand", die mit Zeichenwerkzeugen ausgestattet ist, mit denen man das Erscheinungsbild des Steuerelements beeinflussen kann. Als Zeichenwerkzeuge stehen die Methoden der Graphics-Klasse zur Verfügung. Mit den Zeichenwerkzeugen zeichnen Sie einen schwarzen Rahmen um das Steuerelement und füllen diesen mit der Farbe des selektierten Textes. Das Steuerelement bekommt auch eine Aufgabe – es soll die Farbauswahl aufrufen, wenn der Anwender darauf klickt. Es wird also wie ein PushButton funktionieren, verwendet aber die Methoden der Graphics-Klasse, um sein Aussehen zu steuern.

So fügen Sie das Farbe-Steuerelement hinzu:

1. Klicken Sie auf das StaticText-Objekt, das als Beschriftung für das Größe-Menü dient und duplizieren Sie es.
2. Ziehen Sie das Duplikat rechts neben den Unterstrichen-Knopf und richten Sie es mit der Basis-Linie der anderen Objekte aus. Ändern Sie seine Text-Eigenschaft auf **Farbe:**, setzen Sie die Left-Eigenschaft auf 435 und seine Width-Eigenschaft auf 40.
3. Draggen Sie ein **Canvas**-Steuerelement  aus der Steuerelemente-Palette rechts neben das **StaticText**-Objekt (ignorieren Sie, dass das Canvas ein wenig zu groß ist) und stellen Sie mit dem Eigenschaften-Fenster folgende Werte dafür ein:

Eigenschaft	Wert
Name	KnopfFarbe
Left	482
Top	4
Width	57
Height	16

Die nächsten Schritte zeichnen einen schwarzen Rahmen und füllen das Canvas-Steuerelement mit der Default-Textfarbe.

1. Expandieren Sie die Event-Handler von **KnopfFarbe** im Code-Editor und selektieren Sie den Paint-Event. Eine schnellere Möglichkeit ist, **KnopfFarbe** zu selektieren und **alt-Tab** zu drücken.

   Der **Paint**-Event wird immer dann ausgeführt, wenn REALbasic bemerkt, dass das Canvas-Steuerelement neu gezeichnet werden muss.
   Sie werden bemerken, dass der Methode für den **Paint**-Event ein Parameter **g as Graphics** übergeben wird. Sie verwenden diesen Parameter, um Zugriff auf die Zeichenwerkzeuge der Graphics-Klasse zu bekommen.

2. Geben Sie folgenden Code in den Paint-Event ein:

```
g.ForeColor=rgb(0,0,0) // schwarz
g.DrawRect(0,0,g.Width-1,g.Height-1)
g.ForeColor=TextFeld.SelTextColor
g.FillRect(1,1,g.Width-2,g.Height-2)
```

   Jede Zeile greift auf eine Methode oder Eigenschaft der **Graphics**-Klasse zu. Die **ForeColor**-Eigenschaft legt die Farbe fest, die für darauf folgenden Aufrufe von Methoden der Graphics-Klasse verwendet wird. Die **RGB**-Funktion verwendet das Rot-Grün-Blau-Farbmodell, um der **ForeColor**-Eigenschaft eine Farbe zuzuweisen. Ihre Parameter sind die Farbanteile für Rot, Grün und Blau. Die **ForeColor**-Eigenschaft selbst zeichnet nichts.

   Die erste Code-Zeile setzt **ForeColor** auf Schwarz und die nächste Zeile zeichnet einen schwarzen Rahmen um die Kanten des Steuerelements. Die vier Parameter sind die linken und oberen Koordinaten des zu zeichnenden Rechtecks und die Breite und Höhe des Rechtecks. Die Eigenschaften **Width** und **Height** liefern die aktuelle Breite und Höhe der Zeichenregion zurück. Es ist besser, **Width** und **Height** zu verwenden, als absolute Werte in den Code zu schreiben. Sollte sich die Größe des Steuerelements einmal ändern, müssen Sie diese Code-Zeilen nicht anpassen.

   Die nächsten beiden Zeilen setzen die **ForeColor**-Eigenschaft auf die in TextFeld aktuelle Textfarbe und füllen den Inhalt des Canvas-Steuerelements mit dieser Farbe.

Der nächste Schritt besteht darin, das Verhalten des Canvas-Steuerelements dem eines PushButtons nachzuempfinden. Sie verwenden dazu den **MouseDown**-Event-Handler des Canvas-Steuerelements. Für unsere Zwecke funktioniert er wie der **Action**-Event-Handler des **BevelButton**-Steuerelement. Er wird ausgeführt, wenn der Anwender in das **Canvas**-Steuerelement klickt. Sie werden bemerken, dass die Koordinaten des Mausklicks zurückgeliefert werden, Sie diese jedoch nicht benutzen müssen.

3. Selektieren Sie den **MouseDown**-Event-Handler des **Canvas**-Steuerelements und fügen Sie folgenden Code ein:

```
Dim c as Color
Dim b as Boolean
c=rgb(255,255,255) // Default-Farbe
b=SelectColor(c,"Wähle eine Textfarbe")
If b then
    Me.Graphics.ForeColor=c
    Me.Graphics.FillRect(1,1,Me.Graphics.Width-2,Me.Graphics.Height-2)
    TextFeld.SelTextColor=c
End if
```

Die **SelectColor**-Funktion ruft die Farbauswahl auf. Sie erwartet zwei Parameter, eine Farbe und einen Textstring, der innerhalb der Farbauswahl dargestellt wird. Die Farbe, die Sie an **SelectColor** übergeben, steuert das Aussehen des Farbkreises, der anfangs im Dialog erscheint. **SelectColor** liefert einen boole'schen Wert, der **True** ist, wenn der Anwender auf **OK** geklickt hat und **False**, wenn er die Dialogbox mit **Abbrechen** verlassen hat.

Wenn der Anwender auf **OK** geklickt hat, wird die gewählte Farbe in **c** zurückgeliefert.

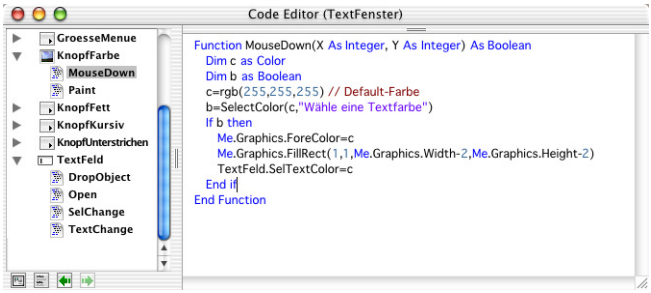
Der Wert von **c**, der von **SelectColor** zurückgeliefert wird, unterscheidet sich von dem, der übergeben wurde (vorausgesetzt, der Anwender hat eine andere Farbe gewählt). Dies ist möglich, da der **Color**-Parameter als Referenz und nicht als Wert übergeben wird. Deshalb ist **SelectColor** in der Lage, den Wert zu ändern und die Referenz zurückzuliefern.

Sie können auch in eigenen Methoden die Übergabe als Referenz verwenden, indem Sie das REALbasic-Schlüsselwort **ByRef** einsetzen. Weitere Informationen dazu finden Sie in der Sprachreferenz.

Das Steuerelement aktualisieren

Sie haben vermutlich schon damit gerechnet, dass Sie nun noch Code in das **SelChange**-Event von TextFeld einfügen müssen, um die Farbe im Canvas-Steuerelement zu aktualisieren, wenn der Cursor auf ein Zeichen bewegt wird, das eine andere Farbe besitzt.

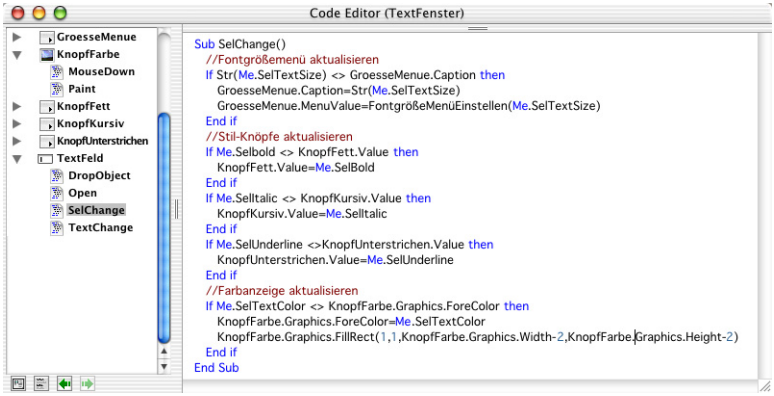
1. Klicken Sie im Code-Editor auf das zweite Icon unter dem Browser-Bereich  , welches „Verstecke leere Methoden“ genannt wird.
   Der Browser zeigt nun nur noch Methoden an, die Code enthalten. Dies macht die Navigation zu Methoden, die modifiziert werden müssen, einfacher.



2. Klicken Sie auf den **SelChange**-Event des **TextFeld**-Objekts und ergänzen Sie den vorhandenen Code folgendermaßen:

```
//Farbanzeige aktualisieren
If Me.SelTextColor <> KnopfFarbe.Graphics.ForeColor then
    KnopfFarbe.Graphics.ForeColor=Me.SelTextColor
    KnopfFarbe.Graphics.FillRect(1,1,KnopfFarbe.Graphics.Width-2,KnopfFarbe.Graphics.Height-2)
End if
```

Der Code-Editor sollte nun so aussehen:



Der Code folgt folgender Logik: Wenn die Farbe des vorhandenen Textes sich von der aktuellen ForeColor-Eigenschaft des Canvas-Steuerelements unterscheidet, wird die ForeColor-Eigenschaft aktualisiert und die FillRect-Methode der Graphics-Klasse verwendet, um das Innere des Canvas-Steuerelements neu zu zeichnen.

Das Farbe-Steuerelement testen

Nachdem der benötigte Code an seinem Platz ist, können Sie nun ausprobieren, ob alles funktioniert:

1. Wählen Sie **Debug/Programm starten** und geben Sie etwas Text in den TextEditor ein.

2. Selektieren Sie etwas Text und versuchen Sie, dessen Farbe zu ändern.

3. Beenden Sie Ihr Programm, um zur Entwicklungsumgebung zurückzukehren.

Rückblick

In diesem Kapitel haben Sie die Steuerelemente **StaticText**, **Separator**, **BevelButton** und **Canvas** kennengelernt.

8. Erzeugen von dynamischen Menüs

In diesem Kapitel werden Sie lernen, wie man ein Menü erzeugt, dessen Einträge erst zur Laufzeit festgelegt werden. Sie werden ein Schrift-Menü programmieren, das eine Liste aller installierter Zeichensätze anzeigt. Im Gegensatz zum Größe-Menü können Sie die Einträge des Schrift-Menüs nicht im voraus festlegen, da Sie nicht wissen, welche Schriften der Anwender auf seinem Rechner installiert hat.

Los geht's

Suchen Sie die REALbasic-Projektdatei, die Sie am Ende des letzten Kapitels gespeichert haben (**TextEditor-Kapitel7**). Starten Sie REALbasic und öffnen Sie das Projekt. Falls nötig, verwenden Sie die mitgelieferte Datei **TextEditor-Kapitel7**, die Sie im Ordner **Tutorial Projekt** finden.

Implementierung des Schrift-Menüs

Das Schrift-Menü verwendet die gleichen Techniken, die Sie in den Kapiteln zuvor kennengelernt haben. Der feine Unterschied ist, dass Sie eine Methode programmieren müssen, die die Namen der installierten Zeichensätze in ein Array einliest. Diese Methode wird ausgeführt, wenn ein Dokumentfenster geöffnet wird.

Zuerst fügen Sie das Schrift-Menü in den Kopfbereich von TextFenster ein:

1. Draggen Sie ein **StaticText**-Steuerelement Aa aus der Steuerelemente-Palette links in den Kopf-Bereich von Text-Fenster und richten Sie die Grundlinie mit den Grundlinien der anderen **StaticText**-Steuerelemente aus, wie in folgender Abbildung gezeigt:

2. Ändern Sie im Eigenschaften-Fenster das StaticText-Steuerelement wie folgt:

Eigenschaft	Wert
Left	7
Top	4

Eigenschaft	Wert
Width	45
Height	16
Text	Schrift:
TextAlign	Rechts
TextFont	System
TextSize	0
Bold	True

3. Draggen Sie ein **BevelButton**-Steuerelement  aus der Steuerelemente-Palette rechts neben den Text „Schrift:“ und richten Sie seine obere Kante mit den anderen Steuerelementen aus.
4. Ändern Sie im Eigenschaften-Fenster die Eigenschaften des BevelButtons wie folgt:

Eigenschaft	Wert
Name	SchriftMenue
Left	55
Top	4
Width	140
Height	16
Caption	(leer lassen)
CaptionAlign	Zentriert
HasMenu	Normales Menü
TextFont	System
TextSize	0

5. Speichern Sie Ihr Projekt unter **TextEditor-Kapitel8**.

Aufbauen des Schrift-Menüs

Die Menüeinträge des Schrift-Menüs werden generiert, wenn der Benutzer eine neue Instanz von TextFenster öffnet. Daher müssen Sie den notwendigen Programmcode in den Open-Event-Handler von **SchriftMenue** einbauen.

1. Selektieren Sie das Steuerelement **SchriftMenue** im TextFenster und drücken Sie alt-Tab.
   Es öffnet sich der Code-Editor für TextFenster und der **Action**-Event von SchriftMenue ist selektiert.
   Falls der **Action**-Event von SchriftMenue nicht selektiert ist, klicken Sie am unteren Rand des Browsers auf das Icon  , um die leeren Methoden anzeigen zu lassen.
2. Selektieren Sie den **Open**-Event-Handler und fügen Sie dort folgenden Code ein:

```
Dim i, nFonts as Integer
nFonts=FontCount-1
For i=0 to nFonts
   me.AddRow Font(i)
Next
me.Caption=TextFeld.SelTextFont
```

Die **FontCount**-Funktion liefert die Anzahl der installierten Zeichensätze. Die **Font**-Funktion liefert den Namen des **i**-ten Zeichensatzes. Die **AddRow**-Methode der **BevelButton**-Klasse fügt dem Menü einen neuen Eintrag hinzu und erwartet den Text des Menüeintrags als Parameter. Die **For**...**Next**-Schleife wird so oft durchlaufen, bis alle Zeichensatznamen hin-

zugefügt wurden. Da Menüeinträge mit Null beginnend durchnummeriert werden, läuft die Schleife von Null bis **FontCount-1** und nicht von **1** bis **FontCount**.

Die letzte Zeile setzt den Defaultwert der Caption-Eigenschaft des Schrift-Menüs auf den Default-Font von TextFeld, also auf den Wert, den die **TextFont**-Eigenschaft von TextFeld hat.

Verwalten des Schrift-Menüs

Das Schrift-Menü muss die Schriftart des ausgewählten Textes auf die Schriftart ändern, die der Anwender im Popup ausgewählt hat. Dies wird mit der **SelTextFont**-Eigenschaft von TextFeld erledigt. Im Popup soll ferner ein Häkchen neben der ausgewählten Schrift erscheinen.

1. Expandieren Sie den **Action**-Event für **SchriftMenue** im TextFenster-Code-Editor.
2. Geben Sie folgenden Code ein:

```
Me.Caption=Me.List(Me.MenuValue)
TextFeld.SelTextFont=SchriftMenue.Caption
```

Die **MenuValue**-Eigenschaft ist die *Nummer* des selektierten Menüeintrags. Die **List**-Methode liefert den Text des zur Nummer gehörenden Menüeintrags. Die erste Zeile setzt die **Caption**-Eigenschaft, also den Text, der im Popup angezeigt wird, auf den Zeichensatz, den der Anwender ausgewählt hat.

Die zweite Zeile weist der **SelTextFont**-Eigenschaft von **TextFeld** (und damit dem aktuell selektierten Text) die gewählte Schrift zu. Neu eingegebener Text erscheint in der ausgewählten Schrift.

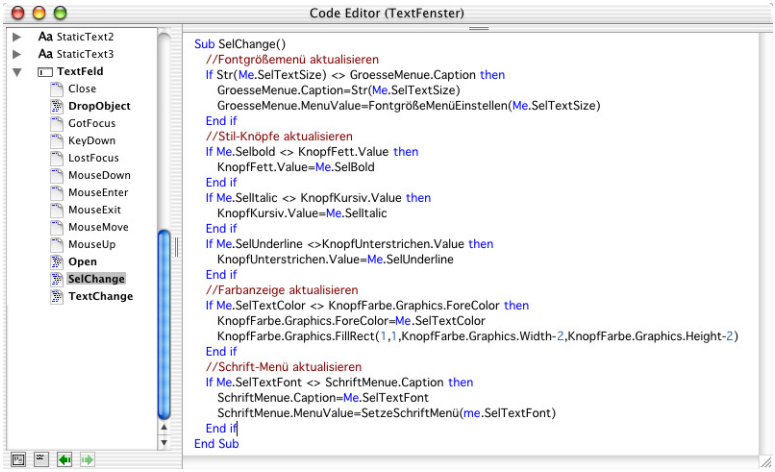
Aktualisieren des Schrift-Menüs

Der letzte Schritt besteht darin, das Schriftmenü zu aktualisieren, wenn der Cursor durch den Text bewegt wird. Da das Bewegen des Cursors die Textselektion ändert, verwenden Sie hierzu den **SelChange**-Event von TextFeld.

1. Expandieren Sie den **TextFeld**-Eintrag im Code-Editor von TextFenster und selektieren Sie den **SelChange**-Event.
2. Fügen Sie an dessen Ende folgenden Code ein:

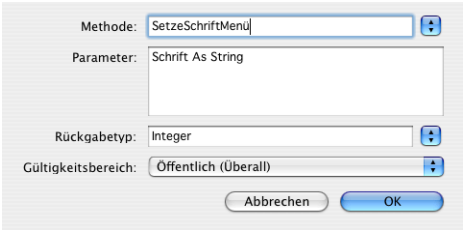
```
//Schrift-Menü aktualisieren
If Me.SelTextFont <> SchriftMenue.Caption then
    SchriftMenue.Caption=Me.SelTextFont
    SchriftMenue.MenuValue=SetzeSchriftMenü(me.SelTextFont)
End if
```

Die SelChange-Methode sollte nun etwa so aussehen:



Der **If**-Befehl überprüft, ob der aktuelle Zeichensatz von dem im Schrift-Menü abweicht. Ist dies der Fall, stellt die nächste Zeile die **Caption**-Eigenschaft neu ein. Die zweite Zeile ist nötig, um das Häkchen zu aktualisieren, das Sie sehen, wenn Sie das Schrift-Menü aufrufen. Die **MenuValue**-Eigenschaft ist die Nummer des gewählten Zeichensatzes, daher benötigen Sie die fortlaufende Nummer, die zu diesem Zeichensatz gehört. Diese ermittelt die **SetzeSchriftMenü**-Methode. Diese müssen Sie **TextFenster** noch hinzufügen:

1. Wählen Sie, während der Code-Editor von TextFenster das oberste Fenster ist, **Bearbeiten/Neue Methode**.
2. Nennen Sie die Methode **SetzeSchriftMenü**, geben Sie als Parameter **Schrift as String** und als Rückgabetyp **Integer** ein. Die Dialogbox sollte nun so aussehen:



3. Klicken Sie auf **OK**.

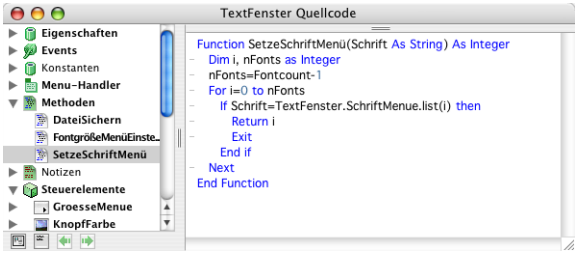
   Beachten Sie, dass es sich bei dieser Methode um eine Funktion handelt, da Sie einen Rückgabetyp und einen Parameter (**Font**) definiert haben.

4. Geben Sie folgenden Code in diese Methode ein:

```
Dim i, nFonts as Integer
nFonts=Fontcount-1
For i=0 to nFonts
    If Schrift=TextFenster.SchriftMenue.list(i) then
        Return i
        Exit
    End if
Next
```

5. Speichern Sie das Projekt.

Die neue Methode sollte folgendermaßen aussehen:



Im **SelChange**-Event-Handler wird die Funktion mit dem Namen des aktuellen Zeichensatzes als Parameter aufgerufen.

Die **For**...**Next**-Schleife vergleicht diesen mit den Namen aller auf dem Rechner installierten Zeichensätze, bis es den passenden findet. Dann gibt es die laufende Nummer **i** zurück und verlässt die Schleife mit dem **Exit**-Befehl.

Nachdem der **SelChange**-Event-Handler die laufende Nummer ermittelt hat, kann er diese der **MenuValue**-Eigenschaft des **SchriftMenues** zuweisen. Dadurch wird das Häkchen im Schrift-Menü korrekt gesetzt.

Testen der Applikation

Nachdem nun die gesamte Werkzeugleiste fertig ist, können Sie die Applikation testen.

1. Wählen Sie **Debug/Programm starten** und experimentieren Sie mit verschiedenen Schriften, Schriftgrößen, Stilen und Farben.
2. Beenden Sie Ihr Programm, um zur Entwicklungsumgebung zurückzukehren.

Rückblick

In diesem Kapitel haben Sie gelernt, wie man Menüeinträge in einem Programm dynamisch erzeugt.

9. Drucken von Text mit Stilinformationen

Nachdem Sie nun Text eingeben können und dessen Schriftart, Schriftstil, Schriftgröße und Farbe frei festlegen können, fehlt eigentlich nur noch die Möglichkeit, den Text auszudrucken.

In diesem Abschnitt geht es darum, die Menüpunkte **Seitenformat** und **Drucken** zu implementieren.

Öffnen Sie die REALbasic-Projektdatei, die Sie am Ende des letzten Kapitels gespeichert haben (**TextEditor-Kapitel8**). Falls nötig, verwenden Sie die mitgelieferte Datei **TextEditor-Kapitel8**, die Sie im Ordner **Tutorial Projekt** finden.

Hinzufügen der Menüpunkte Seitenformat und Drucken

Um diese Menüpunkte einzurichten, gehen Sie folgendermaßen vor:

1. Öffnen Sie den Menüeditor und selektieren Sie den leeren Eintrag am Ende des **Ablage**-Menüs (Windows: des **Datei**-Menüs).
2. Tragen Sie im Eigenschaften-Fenster als Text-Eigenschaft **Seitenformat…** und als **Name**-Eigenschaft **AblageSeitenformat** (ohne die drei Punkte am Schluss) ein.
3. Deaktivieren Sie die **AutoEnable**-Eigenschaft.
4. Selektieren Sie den leeren Menüeintrag am Ende des **Ablage**-Menüs (Windows: des **Datei**-Menüs), geben Sie als **Text**-Eigenschaft **Drucken…** und als Name-Eigenschaft **AblageDrucken** (wieder ohne die drei Punkte) ein.
5. Weisen Sie der **CommandKey**-Eigenschaft den Buchstaben **P** zu.
6. Deaktivieren Sie die **AutoEnable**-Eigenschaft.
7. Positionieren Sie die beiden neuen Menüpunkte zwischen den **Sichern unter**- und **Beenden**-Menüpunkten.
8. Selektieren Sie den leeren Menüeintrag am Ende des **Ablage**-Menüs (Windows: des **Datei**-Menüs) und geben Sie als **Text**-Eigenschaft ein Minus („**-**“) ein. Dadurch erzeugen Sie einen Separator zwischen Gruppen von Menüeinträgen.
9. Draggen Sie den Separator zwischen die **Sichern unter**- und **Seitenformat**-Menüpunkte.
10. Erzeugen Sie einen zweiten Separator und draggen Sie ihn zwischen die **Drucken**- und **Beenden**-Menüpunkte.
11. Schließen Sie den Menü-Editor.
12. Speichern Sie das Projekt unter **TextEditor-Kapitel9**.

Aktivieren der Menüpunkte Seitenformat… und Drucken…

Diese Menüpunkte sollen immer dann aufrufbar sein, wenn ein Dokumentfenster geöffnet ist. Deshalb werden sie im Code-Editor für TextFenster aktiviert:

1. Klicken Sie im Code-Editor-Browser von TextFenster auf das Dreieck vor dem **Events**-Eintrag.
2. Selektieren Sie den **EnableMenuItems**-Event und ergänzen Sie den vorhandenen Code um folgende Anweisungen:

```
AblageSeitenformat.Enable
AblageDrucken.Enable
```

Verwalten des Seitenformat-Menüpunkts

Damit Sie die Einstellungen verwenden können, die der Anwender im Seitenformat-Dialog vornimmt, benötigen Sie ein **PrinterSetup**-Objekt. Dieses Objekt hat die Eigenschaft **SetupString**, die viele dieser Einstellungen speichert. Der erste Schritt besteht darin, diese Eigenschaft im Code-Editor für TextFenster anzulegen.

1. Rufen Sie den Menüpunkt **Bearbeiten/Neue Eigenschaft** auf und geben Sie folgendes in die erscheinende Eigenschaften-Definitionsbox ein: **Seitenformat as String**. Wählen Sie als Gültigkeitsbereich **Geschützt (Textfenster und Unterklassen)**.
2. Klicken Sie auf **OK**, um das Fenster zu schließen.

Als nächstes werden Sie den Menu-Handler für den Seitenformat-Menüpunkt programmieren.

1. Rufen Sie **Bearbeiten/Neuer Menu-Handler** auf und wählen Sie aus dem Popup-Menü **AblageSeitenformat** aus.
2. Geben Sie folgenden Programmcode für den Seitenformat-Menü-Handler ein:

```
Dim ps as PrinterSetup
ps=New PrinterSetup
If Seitenformat<>"" then
    ps.setupstring=Seitenformat
End if
If ps.PageSetupDialog then
    Seitenformat=ps.SetupString
end if
```

Die PrinterSetup-Eigenschaft, **SetupString**, speichert die Einstellungen, die der Anwender in der Seitenformat-Dialogbox vornimmt. Die zweite **If**-Anweisung zeigt die Seitenformat-Dialogbox an. Wenn der Anwender im Seitenformat-Dialog die **OK**-Taste betätigt, liefert **PageSetupDialog True** zurück und **SetupString** wird mit der **Seitenformat**-Eigenschaft belegt. Der Menu-Handler verwendet die **Seitenformat**-Eigenschaft, um die Einstellungen des Anwenders zu speichern.

Verwalten des Drucken…-Menüpunkts

Zum Drucken von Text mit Stilinformationen verwenden Sie ein Objekt des Typs **StyledTextPrinter**. Dieses verfügt über die **DrawBlock**-Eigenschaft, mit deren Hilfe der Text zu Papier gebracht wird.

Der **Drucken…**-Menu-Handler wird folgendermaßen implementiert:

1. Rufen Sie den Menüpunkt **Bearbeiten/Neuer Menu-Handler** auf und wählen Sie aus dem Popup-Menü den Eintrag **AblageDrucken**.
2. Geben Sie folgenden Programmcode für den Menu-Handler ein:

```
Dim stp as StyledTextPrinter
Dim g as Graphics
Dim ps as PrinterSetup
Dim Seitenbreite as Integer
Dim Seitenlänge as Integer
ps=New PrinterSetup
If Seitenformat<>"" then //Seitenformat enthält Werte
    ps.SetupString=Seitenformat
    Seitenbreite=ps.width-36
    Seitenlänge=ps.height-36
    //Drucken-Dialog mit den Werten aus Seitenformat öffnen
    g=openPrinterDialog(ps)
else
    g=openPrinterDialog()//Dialog ohne Seitenformat-Angaben öffnen
    Seitenbreite=72*7.5 //Defaultbreite und -höhe
    Seitenlänge=72*9
End if
If g<>Nil then //Der Anwender hat den Drucken-Dialog nicht abgebrochen
    stp=TextFeld.StyledTextPrinter(g,Seitenbreite-48)
```

```
    Do until stp.eof
       stp.drawBlock 36,36,Seitenlänge-48
       if not stp.eof then //gibt es noch zu druckenden Text?
          g.NextPage
       end if
    Loop
end if
```

Der Menu-Handler verwendet die **StyledTextPrinter**-Methode der **EditField**-Klasse, um ein **StyledTextPrinter**-Objekt (**stp**) zu erzeugen. Falls der Anwender im Seitenformat-Dialog Einstellungen vorgenommen hat, ist die **Seitenformat**-Eigenschaft nicht Null und ihre Werte können zum Drucken verwendet werden. Die Eigenschaften **Width** und **Height** des **PrinterSetup**-Objekts sind die Höhe und Breite des gesamten bedruckbaren Bereichs, wie er in der Seitenformat-Dialogbox festgelegt ist. Normalerweise verwendet ein Textdokument mit Stilinformationen zusätzliche Ränder links, rechts, oben und unten. Daher wurden die **Width**- und **Height**-Eigenschaften entsprechend verkleinert. Sie können natürlich auch andere Werte für das Seitenformat wählen.

Wenn der Anwender den Seitenformat-Dialog nicht aufruft, werden die Vorgabewerte für Höhe und Breite des bedruckbaren Bereichs verwendet.

Da das TextFeld mehr als eine Textseite beinhalten kann, müssen Sie das Drucken mehrerer Seiten unterstützen. Die boole'sche Eigenschaft **EOF** (end of file) eines **StyledTextPrinter**-Objekts ist so lange **False**, so lange noch Text zu drucken ist. Die **Do Loop** Schleife wird deshalb so lange ausgeführt, bis **EOF True** ist. Sie enthält einen Aufruf der **DrawBlock**-Methode, welche einen Textblock auf die Seite druckt.

```
stp.drawBlock 36,36, Seitenlänge-48
```

Die ersten beiden Parameter legen den Abstand der linken oberen Ecke des Blocks von der linken oberen Ecke des im Seitenformat definierten bedruckbaren Bereichs fest.

Als dritter Parameter wird die Höhe des Textblocks übergeben. (Die Breite des Blocks wurde bereits der StyledTextPrinter-Methode als Parameter **Seitenbreite** übergeben).

Sobald Sie einen Textblock gedruckt haben, müssen Sie feststellen, ob weiterer Text gedruckt werden muss. Ist dies der Fall, müssen Sie die **NextPage**-Methode der **Graphics**-Klasse zum Erzeugen einer neuen Seite aufrufen. Dies wird durch die **If**-Anweisung innerhalb von **Do loop** gewährleistet.

Im Beispielcode wurden die an **DrawBlock** übergebenen Parameter passend für eine 8,5x11 Zoll große Seite gewählt. Für andere Seitenformate sollten Sie die Werte von **pageWidth**, **pageHeight** und die Position der linken oberen Ecke des Druckbereichs anpassen.

Testen der Druckfunktion

1. Rufen Sie den Menüpunkt **Debug/Programm starten** auf und geben Sie Text in den Texteditor ein.
2. Weisen Sie dem Text verschiedene Schriftarten, Schriftgrößen und Stile zu.
3. Verwenden Sie die Menüpunkte **Seitenformat…** und **Drucken…**, um die Druckfunktion zu testen.
4. Beenden Sie Ihr Programm, um in die Entwicklungsumgebung zurückzukehren.

Rückblick

In diesem Kapitel haben Sie Ihr Programm um die Funktion erweitert, Texte mit Stilinformationen zu drucken.

10. Kommunikation zwischen Fenstern

In diesem Kapitel erfahren Sie, wie REALbasic-Objekte miteinander kommunizieren. Sie werden lernen, wie man:

- eine Finden- und Ersetzen-Dialogbox realisiert,

- Code schreibt, der die Kommunikation zwischen dem Finden- und Ersetzen-Dialog und Texteditor erlaubt.

Die Finden-Funktion, die Sie einbauen werden, ist eine einfache Routine, die ab Cursorposition bis zum Ende des Textes sucht. Sie wird keine Möglichkeit bieten, wieder an den Textanfang zurückzugehen und die Suche von dort aus erneut zu starten. Sie können diese Funktionen allerdings später selbst einbauen. Die hier realisierte Finden-Funktion unterscheidet nicht zwischen Groß- und Kleinschreibung.

Los geht's

Öffnen Sie die REALbasic-Projektdatei, die Sie am Ende des letzten Kapitels gespeichert haben (**TextEditor-Kapitel9**). Falls nötig, verwenden Sie die mitgelieferte Datei **TextEditor-Kapitel9**, die Sie im Ordner **Tutorial Projekt** finden.

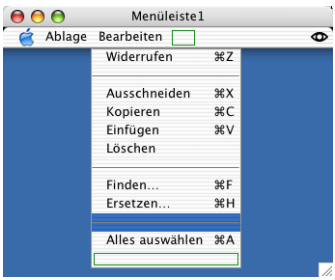
Implementieren einer Finden-Dialogbox

Sie sind bereits damit vertraut, wie Menüpunkte hinzugefügt, aktiviert und ihre Menu-Handler implementiert werden. In diesem Kapitel ist neu, dass die Dialogbox, die durch einen Menüeintrag aufgerufen und angezeigt wird, mit einem anderen Fenster der Applikation kommunizieren muss.

Sie beginnen damit, einen **Finden**-Menüpunkt in das **Bearbeiten**-Menü einzubauen.

Erzeugen des Menüeintrags

1. Führen Sie einen Doppelklick auf **Menüleiste1** im Projekt-Fenster aus.
2. Selektieren Sie im Menü-Editor das **Bearbeiten**-Menü.
3. Selektieren Sie den leeren Menüeintrag am Ende des Bearbeiten-Menüs und geben Sie **Finden…** als seine **Text**-Eigenschaft ein. Als **Name**-Eigenschaft tragen Sie **BearbeitenFinden** (ohne die drei Punkte am Schluss!) ein.
4. Geben Sie **F** als **CommandKey**-Eigenschaft an. Deaktivieren Sie die **AutoEnable**-Eigenschaft.
5. Selektieren Sie den leeren Menüeintrag am Ende des Bearbeiten-Menüs und geben **Ersetzen…** als seine **Text**-Eigenschaft ein. Als **Name**-Eigenschaft tragen Sie **BearbeitenErsetzen** (ohne die drei Punkte am Schluss!) ein.
6. Geben Sie **H** als **CommandKey**-Eigenschaft an. Deaktivieren Sie die **AutoEnable**-Eigenschaft.
7. Selektieren Sie den leeren Menüeintrag am Ende des Edit-Menüs und geben Sie als **Text**-Eigenschaft ein Minus „**-**“ an. Damit erzeugen Sie eine Trennlinie.
8. Draggen Sie die Trennlinie zwischen die Menüeinträge **Löschen** und **Alles auswählen**.
9. Draggen Sie den Menüpunkt **Alles auswählen** an das untere Ende, fügen Sie einen weiteren Separator ein und platzieren Sie diesen zwischen **Ersetzen** und **Alles auswählen**.
   Das **Bearbeiten**-Menü sollte nun folgendermaßen aussehen:



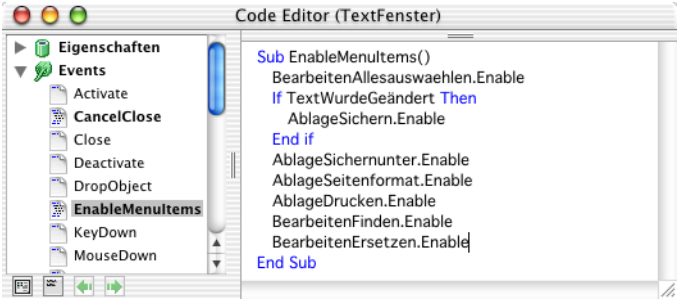
10. Schließen Sie den Menü-Editor und speichern Sie Ihr Projekt unter **TextEditor-Kapitel10**.

Aktivieren des Finden- und Ersetzen-Menüeintrags

Der **Finden**-Menüpunkt soll nur dann aktiv sein, wenn ein Textfenster geöffnet ist. Deshalb wird er im Code-Editor für **TextFenster** aktiviert.

Um den Menüeintrag zu aktivieren, müssen Sie folgendes tun:

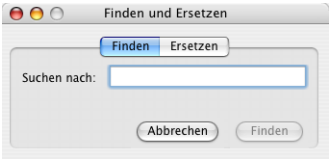
1. Öffnen Sie den Code-Editor für **TextFenster** im Projekt-Fenster.
2. Selektieren Sie im Browser den **EnableMenuItems** Event-Handler.
3. Fügen Sie folgenden Code an das Ende der Methode an:
   ```
   BearbeitenFinden.Enable
   BearbeitenErsetzen.Enable
   ```
   Der Code-Editor sollte folgendermaßen aussehen:

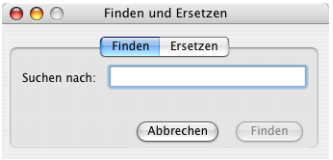


4. Schließen Sie den Code-Editor und speichern Sie Ihr Projekt.

Erzeugen der Finden- und Ersetzen-Dialogbox

Die nächste Aufgabe ist das Erzeugen der Dialogbox. Sie werden hierzu eine einzelne Dialogbox erzeugen, die für die Suchen- und für die Ersetzen-Funktion zuständig ist. Dazu werden Sie ein **TabPanel**-Steuerelement verwenden, das dem Anwender gestattet, eine der beiden Funktionen auszuwählen, nachdem die Dialogbox geöffnet wurde. Die Dialogbox soll am Ende folgendermaßen aussehen:





Beginnen Sie damit, Ihrem Projekt ein neues Fenster hinzufügen.

1. Holen Sie das Projekt-Fenster nach oben. Wählen Sie **Ablage/Neues Fenster** (Windows: **Datei/Neues Fenster**).
   REALbasic erzeugt im Projekt ein neues Fenster namens **Fenster1**.
2. Ändern Sie den Namen des Fensters im Eigenschaften-Fenster auf **FindenFenster**.
3. Ändern Sie die Title-Eigenschaft auf **Finden und Ersetzen**.
4. Ändern Sie Breite und Höhe: **Width: 340**, **Height: 140**.
5. Deselektieren Sie die Eigenschaften **GrowIcon** und **ZoomIcon**.
   Diese Eigenschaften werden deselektiert, weil FindenFenster eine Dialogbox fester Größe sein soll.

Mit folgenden Schritten fügen Sie dem leeren Fenster die Steuerelemente hinzu:

1. Draggen Sie aus der Steuerelemente-Palette ein **TabPanel**-Steuerelement  in das **FindenFenster**.

2. Setzen Sie die Eigenschaften **Left**, **Top**, **Width** und **Height** des TabPanels auf die Werte **8**, **6**, **321** und **124**.
3. Klicken Sie auf den Karteireiter mit den drei Punkten, um den Panel-Editor aufzurufen.
4. Klicken Sie auf **Tab 0** und dann auf den **Bearbeiten**-Knopf.
5. Ändern Sie seinen Namen in **Finden** und klicken Sie auf **OK**.
6. Klicken Sie auf **Tab 1** und dann auf den **Bearbeiten**-Knopf.
7. Ändern Sie den Namen in **Ersetzen** und klicken Sie auf **OK**.
8. Klicken Sie auf **OK**, um den Panel-Editor zu beenden.

Als nächstes werden die benötigten Steuerelemente auf der Finden-Karteikarte platziert.

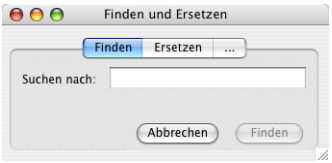
1. Klicken Sie auf den **Finden**-Karteireiter. Draggen Sie ein **StaticText**-Steuerelement  in die linke obere Ecke der Karteikarte. Dieses Steuerelement nimmt die Beschriftung des Eingabefeldes auf.
2. Setzen Sie die Eigenschaften Left, Top, Width und Height auf **21**, **48**, **84** und **16** und ändern Sie die **Text**-Eigenschaft auf **Suchen nach:**.
3. Draggen Sie aus der Steuerelemente-Palette ein **EditField**-Steuerelement  rechts neben das **StaticText**-Steuerelement und weisen ihm folgende Eigenschaften zu:

Eigenschaft	Wert
Name	FindenText
Left	112
Top	43
Width	204
Height	22

4. Ziehen Sie einen **PushButton**  aus der Steuerelemente-Palette in die rechte untere Ecke der Karteikarte.
5. Selektieren Sie den **PushButton** und rufen Sie **Bearbeiten/Duplizieren** auf, um den zweiten Knopf zu erzeugen. Weisen Sie den PushButtons folgende Eigenschaften zu (der rechte ist der **Finden**-Knopf):

Eigenschaft	Abbrechen-Knopf	Finden-Knopf
Name	AbbrechenKnopf	FindenKnopf
Left	140	243
Top	101	101
Width	85	70
Caption	Abbrechen	Finden
Default	Nicht aktiv	Aktiv
Cancel	Aktiv	Nicht aktiv
Enabled	Aktiv	Nicht aktiv

Die Finden-Karteikarte sollte jetzt so aussehen:



Als nächstes müssen Sie die Steuerelemente der **Ersetzen**-Karteikarte erzeugen. Die Ersetzen-Karteikarte entspricht der Finden-Karteikarte mit dem Unterschied, dass sie zusätzlich ein Eingabefeld für den Ersatztext samt Feldbeschriftung enthält.

1. Klicken Sie auf den **Ersetzen**-Karteireiter des **TabPanel**-Steuerelements.

   Dies blendet die Steuerelemente des **Finden**-Panels aus und erlaubt Ihnen, einen neuen Satz von Steuerelementen zu platzieren, die nur angezeigt werden, wenn der Anwender auf den **Ersetzen**-Karteireiter klickt.

2. Draggen Sie ein **StaticText**-Steuerelement  in den linken oberen Bereich. Das Steuerelement wird als Beschriftung für den **Finden**-Eintrag im **Ersetzen**-Panel dienen.

3. Setzen Sie die Eigenschaften **Left**, **Top**, **Width** und **Height** auf **21**, **48**, **84** und **16** und ändern Sie die **Text**-Eigenschaft auf **Suchen nach:**.

4. Draggen Sie aus der Steuerelemente-Palette ein **EditField**-Steuerelement  rechts neben das **StaticText**-Steuerelement und weisen ihm folgende Eigenschaften zu:

Eigenschaft	Wert
Name	SuchText
Left	112
Top	43
Width	204
Height	22

5. Duplizieren Sie **StaticText** und **EditField** und verschieben Sie diese nach unten.

6. Setzen Sie die Eigenschaften **Left**, **Top**, **Width** und **Height** des neuen **StaticText**-Steuerelements auf **21**, **73**, **84** und **16** und ändern Sie die Text-Eigenschaft in **Ersetze durch:**.

7. Setzen Sie die Eigenschaften des **EditField**-Steuerelements auf folgende Werte:

Eigenschaft	Wert
Name	ErsetzeText
Left	112
Top	70
Width	204
Height	22

8. Ziehen Sie als nächstes ein **PushButton**-Steuerelement aus der Steuerelementepalette in die rechte untere Ecke der Karteikarte.

9. Duplizieren Sie den PushButton und ziehen Sie die Kopie links neben das Original.

10. Weisen Sie den PushButtons folgende Eigenschaften zu (der rechte ist der **Ersetzen**-Knopf):

Eigenschaft	Abbrechen	Ersetzen
Name	ErsetzenAbbrechen	ErsetzenKnopf
Left	140	243
Top	101	101
Width	85	70
Caption	Abbrechen	Ersetzen

Eigenschaft	Abbrechen	Ersetzen
Default	Nicht aktiv	Aktiv
Cancel	Aktiv	Nicht aktiv
Enabled	Aktiv	Nicht aktiv

Die Ersetzen-Karteikarte sollte nun so aussehen:

Aktionen für jedes Steuerelements festlegen

Jetzt müssen Sie die Aktionen für jedes Steuerelement festlegen.

1. Klicken Sie im Projekt-Fenster auf **FindenFenster** und öffnen Sie den Code-Editor mit alt-Tab.
2. Expandieren Sie den **Steuerelemente**-Eintrag.

   Hier sehen Sie die Namen der Objekte, die Sie gerade in das **FindenFenster** eingebaut haben.
3. Expandieren Sie **FindenText** und klicken Sie auf den **TextChange**-Event-Handler. Dieser wird ausgeführt, wenn der Anwender in der **Finden**-Dialogbox Text eingibt oder editiert. Geben Sie folgende Code-Zeilen ein:

   ```
   If len(Me.Text)>0 then //wenn der Anwender Text eingegeben hat
       FindenKnopf.Enabled=True
   else
       FindenKnopf.Enabled=False
   end if
   ```

   Der **If**-Befehl stellt fest, ob das **FindenText**-Feld Text enthält (die **Me**-Funktion ist eine Referenz auf das Steuerelement, dem der Event-Handler gehört – in diesem Fall **FindenText**). Wenn das der Fall ist, wird der **Finden**-Knopf aktiviert.
4. Expandieren Sie **SuchText** (der **Finden**-Eingabebereich im **Ersetzen**-Bereich) und klicken Sie auf den **TextChange**-Event-Handler. Fügen Sie folgenden Code in den Event-Handler ein:

   ```
   If len(Me.Text)>0 then
       ErsetzenKnopf.Enabled=True
   else
       ErsetzenKnopf.Enabled=False
   end if
   ```

   Dieser Code aktiviert den **Ersetzen**-Knopf auf der **Ersetzen**-Karteikarte.
5. Expandieren Sie **AbbrechenKnopf** und dort **Action**. Geben Sie folgende Zeile ein:

   ```
   Close
   ```

   Damit wird das Fenster geschlossen, indem eine Methode der Window-Klasse aufgerufen wird.
6. Expandieren Sie **ErsetzenAbbrechen** und fügen Sie dort ebenfalls folgende Zeile in den **Action**-Event-Handler ein:

   ```
   Close
   ```
7. Expandieren Sie **FindenKnopf** und klicken Sie dann auf **Action**. Geben Sie folgende Zeilen ein:

   ```
   TextFenster(Window(1)).Finden FindenText.Text,""
   Close
   ```

   Diese Methode ruft eine Methode namens **Finden** auf, die die eigentliche Arbeit verrichtet. Sie übernimmt als Parameter den Text, den der Anwender auf der **Finden**-Karteikarte eingegeben hat.

Der zweite Parameter der Finden-Methode ist der Ersatztext, der für die Finden-Funktion nicht benötigt wird und deshalb als leerer String übergeben wird.

Die **Window**-Funktion wird verwendet, um das **TextFenster** zu bestimmen, in dem gesucht werden soll. Der Ausdruck **Window(1)** verweist auf das zweite Fenster – **Window(0)** ist der Finden und Ersetzen Dialog selbst – daher ist **Window(1)** das oberste Dokumentfenster.

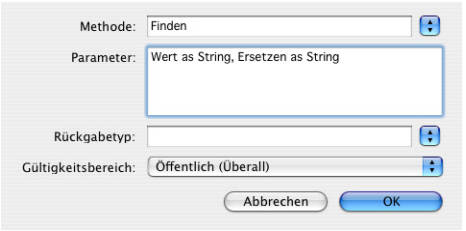
8. Expandieren Sie **ErsetzenKnopf** und klicken Sie auf den **Action**-Event-Handler. Fügen Sie folgenden Code hinzu:

```
TextFenster(Window(1)).Finden SuchText.Text,ErsetzeText.Text
Close
```

Dieser Code übergibt den Inhalt von **ErsetzeText** als zweiten Parameter an die **Finden**-Methode.

Finden-Methode für TextFenster hinzufügen

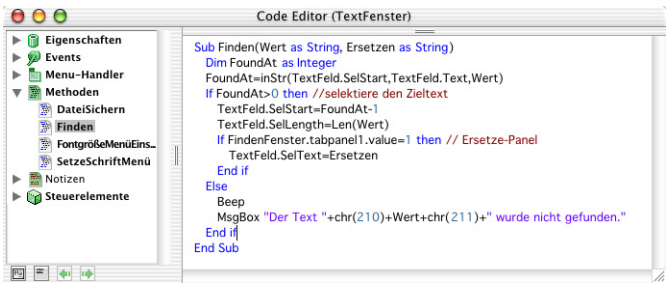
1. Selektieren Sie **TextFenster** im Projekt-Fenster und öffnen Sie mit alt-Tab den Code-Editor.
2. Wählen Sie **Bearbeiten/Neue Methode**, um die Finden-Methode zu erzeugen.
3. Geben Sie **Finden** als Methoden-Namen und **Wert as String, Ersetzen as String** als Parameter ein. Die Dialogbox sollte so aussehen:



4. Klicken Sie auf **OK**, um den Code-Editor der **Finden**-Methode anzuzeigen.
5. Geben Sie folgende Quelltextzeilen in den Code-Editor ein:

```
Dim FoundAt as Integer
FoundAt=inStr(TextFeld.SelStart,TextFeld.Text,Wert)
If FoundAt>0 then //selektiere den Zieltext
    TextFeld.SelStart=FoundAt-1
    TextFeld.SelLength=Len(Wert)
    If FindenFenster.tabpanel1.value=1 then // Ersetze-Panel
       TextFeld.SelText=Ersetzen
    End if
Else
    Beep
    MsgBox "Der Text "+chr(210)+Wert+chr(211)+" wurde nicht gefunden."
End if
```

Der Code-Editor sollte nun so aussehen:



Diese Methode ermittelt die Position des gesuchten Strings (Parameter **Wert**) mit Hilfe der **InStr**-Funktion. **InStr** erwartet drei Parameter: die Startposition, den Text, der durchsucht werden soll und den Text, nach dem gesucht wird. Verläuft die Suche erfolgreich, wird die Fundstelle mit Hilfe der **SelStart**- und **SelLength-**Eigenschaften von TextFeld im Fenster hervorgehoben.

Der zweite **If**-Befehl überprüft, ob der Anwender die Ersetzen-Karteikarte der Dialogbox verwendet hat. (Die **Value**-Eigenschaft eines TabPanel-Steuerelements liefert die Nummer der Karteikarte zurück, wobei die erste Karteikarte die Nummer Null hat). In diesem Fall wird der gefundene Text (dieser entspricht der **SelText**-Eigenschaft) durch den Inhalt des **Ersetzen**-Parameters ersetzt.

Jetzt fehlen noch die Menu-Handler für die **Finden**- und **Ersetzen**-Menüeinträge. Der jeweilige Menu-Handler sorgt dafür, dass die richtige Karteikarte der Dialogbox angezeigt wird.

1. Rufen Sie bei aktivem Code-Editor von TextFenster den Menüpunkt **Bearbeiten/Neuer Menu-Handler** auf.
2. Wählen Sie **BearbeitenFinden** aus dem Popup des Menu-Handler-Dialogs und klicken Sie auf **OK**.
3. Geben Sie folgende Zeile in die Menu-Handler-Methode ein:
   ```
   FindenFenster.Show
   ```
   Show ist eine Methode der **Window**-Klasse. Diese Zeile stellt die Dialogbox dar.
4. Rufen Sie erneut **Bearbeiten/Neuer Menu-Handler** auf, wählen Sie **BearbeitenErsetzen** und **OK**.
5. Fügen Sie folgenden Code in den Menu-Handler des **Ersetzen**-Menüeintrags ein:
   ```
   FindenFenster.Show
   FindenFenster.TabPanel1.Value=1
   ```
   Mit Show wird die Dialogbox angezeigt. Die zweite Zeile sorgt dafür, dass die zweite Karteikarte des **TabPanel**-Steuerelements dargestellt wird (die erste Karteikarte hat die Nummer 0, die zweite die Nummer 1).

Testen der Finden- und Ersetzen-Funktionen

Wählen Sie **Debug/Programm starten**, geben Sie Text ein und testen Sie die Menüpunkte **Finden** und **Ersetzen**.

Sie werden folgenden Schwachpunkt bemerken: Wenn Sie die Karteikarte wechseln, wird Text, den Sie ins **Suchen nach**-Feld eingegeben haben, nicht auf die andere Karteikarte übernommen. Das lässt sich leicht ändern:

1. Kehren Sie in die Entwicklungsumgebung zurück und expandieren Sie das **TabPanel1**-Objekt im Code-Editor von **FindenFenster**.
2. Selektieren Sie den **Change**-Event-Handler.

   Dieser Event wird ausgeführt, wenn der Anwender auf einen Karteireiter klickt.
3. Geben Sie folgenden Code ein:
   ```
   //Wenn das Finden-Feld nicht leer ist, wenn der Anwender
   //zwischen den Panels umschaltet, wird das andere
   //Panel mit dem Finden-Text aktualisiert.
   Select case TabPanel1.Value
   Case 0 //Finden-Panel
       If SuchText.text <> "" then
           FindenText.text=SuchText.Text
       End if
   Case 1 //Ersetzen-Panel
       If FindenText.text <> "" then
           SuchText.text=FindenText.text
       End if
   End select
   ```
   Der **Select Case**-Befehl kopiert abhängig davon, welche Karteikarte gerade angezeigt wird, den Inhalt des „Suchen nach"-Feldes in das „Suchen nach"-Feld der anderen Karteikarte.
4. Speichern Sie Ihr Projekt und testen Sie Ihr Programm.

Rückblick

In diesem Kapitel haben Sie ein TabPanel-Objekt verwendet und erfahren, wie man Objekte erzeugt, die miteinander kommunizieren.

11. Fehlersuche

In diesem Kapitel werden Sie mit dem REALbasic-Debugger arbeitet. Sie werden lernen, wie man:

- Syntax-Fehler findet und behebt,
- den Debugger benutzt, um logische Fehler im Quelltext zu finden,
- Laufzeitfehler behandelt.

Los geht's

Öffnen Sie das Projekt **TextEditor-Kapitel10**. Falls nötig, verwenden Sie die mitgelieferte Datei, die Sie im Ordner **Tutorial Projekt** finden.

Benutzen des Debuggers

Der REALbasic-Debugger unterstützt Sie bei der Fehlersuche und ist einfach zu benutzen. Im Grunde genommen haben Sie den Debugger bereits benutzt, ohne es zu wissen.

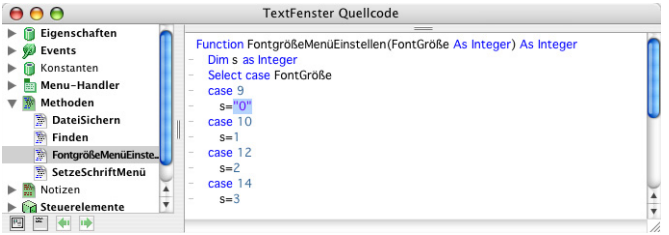
Automatische Debugging-Features

Ein Teil des REALbasic-Debuggers ist bereits dann aktiv, während Sie Quelltextzeilen eintippen. Automatisches Einrücken und farbliches Hervorheben von Schlüsselwörtern machen die Arbeit im Code-Editor übersichtlicher und helfen, Fehler zu vermeiden.

Der Debugger wird auch dann aktiv, wenn Sie **Debug/Programm starten** aufrufen. Dann überprüft REALbasic Ihren gesamten Code auf Syntax-Fehler und stoppt, wenn es einen findet.

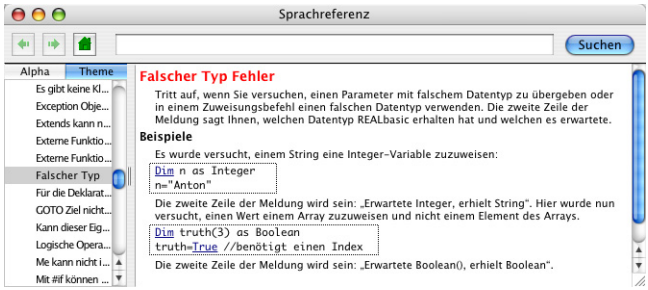
Um die Syntax-Prüfung in Aktion zu erleben, machen Sie einmal folgendes:

1. Öffnen Sie den Code-Editor für **TextFenster** und expandieren Sie den Eintrag **Methoden**.
2. Klicken Sie die **FontgrößeMenüEinstellen**-Methode an, um deren Code anzuzeigen.
3. Ändern Sie die Zeile
   `s=0`
   in
   `s="0"`
   Dadurch ändern Sie den Datentyp der Null von Zahl auf String.
4. Rufen Sie den Menüpunkt **Debug/Programm starten** auf. Es erscheint die Fehlermeldung „Falscher Typ". Die betroffene Programmzeile ist hervorgehoben:



Die Variable s wurde als Integer deklariert, deshalb können ihr nur Zahlen zugewiesen werden.

Wenn Sie auf den **Hilfe**-Knopf rechts neben der Fehlermeldung klicken, wird die Online-Referenz geöffnet und der Eintrag zu diesem Fehler angezeigt.



5. Korrigieren Sie den Fehler, indem Sie die Anführungszeichen wieder löschen.
6. Testen Sie das Programm erneut.

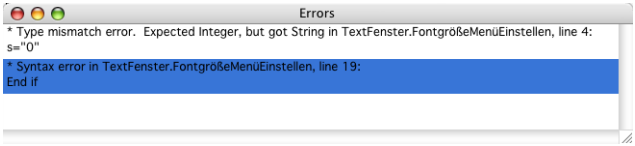
   Da keine weiteren Syntaxfehler auftreten, kann REALbasic Ihr Programm compilieren.

Alle Fehlermeldungen anzeigen

Sie können festlegen, dass alle während eines Compiler-Laufs aufgetretenen Fehler auf einmal angezeigt werden. Standardmäßig verhält sich REALbasic wie oben beschrieben: Der Compiler-Lauf wird beim ersten Fehler unterbrochen. Ob Ihr Quelltext weitere Fehler enthält, erfahren Sie erst beim nächsten Compiler-Lauf.

Wenn Sie ein größeres Projekt bearbeiten ist es wahrscheinlich effizienter, wenn Sie die Option **Alle Compiler-Fehlermeldungen zeigen** aktivieren. Diese finden Sie im REALbasic **Einstellungen**-Dialog unter **Erzeugen-Prozess**.

Ist diese Option aktiviert, werden alle Fehler in einem Fehlerfenster aufgelistet:



Indem Sie einen Doppelklick auf eine Fehlermeldung ausführen, springen Sie zur Fehlerposition im Quelltext.

Suchen logischer Programmfehler

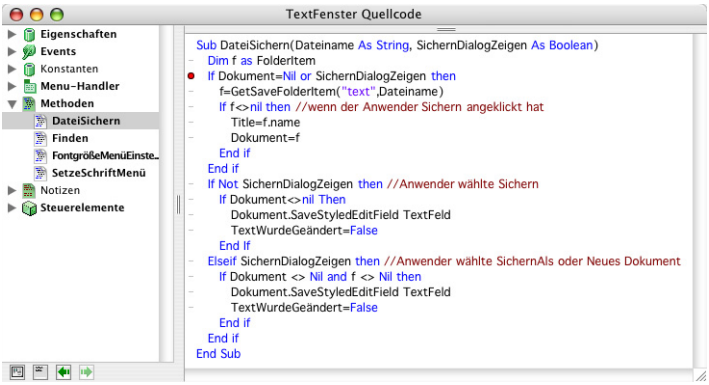
Fehler, die während des Programmablaufs auftreten, sind normalerweise logische Fehler. Um diese Fehler zu finden, müssen Sie REALbasic zunächst mitteilen, wo es suchen soll.

Als erstes müssen Sie im Quelltext Ihres Programms in der Routine, in der Sie den Fehler vermuten, einen *Breakpoint* setzen. An einem Breakpoint wird das Programm gestoppt und der Debugger aufgerufen. Im Debugger können Sie dann die Werte der einzelnen Variablen, Eigenschaften und anderer Parameter überprüfen. Sie können nach unerwarteten, falschen oder undefinierten Werten suchen und entsprechende Korrekturen vornehmen. Sie können ebenso überprüfen, ob eine Methode auch wirklich dann aufgerufen wird, wenn Sie es erwarten.

Breakpoints verändern Ihren Code nicht und funktionieren nur in der REALbasic-Laufzeitumgebung, nicht in Standalone-Programmen. Die folgenden Übungen zeigen, wie Sie Ihr Programm anhalten, die aktuellen Werte von Variablen überprüfen und eine Methode Zeile für Zeile abarbeiten lassen können:

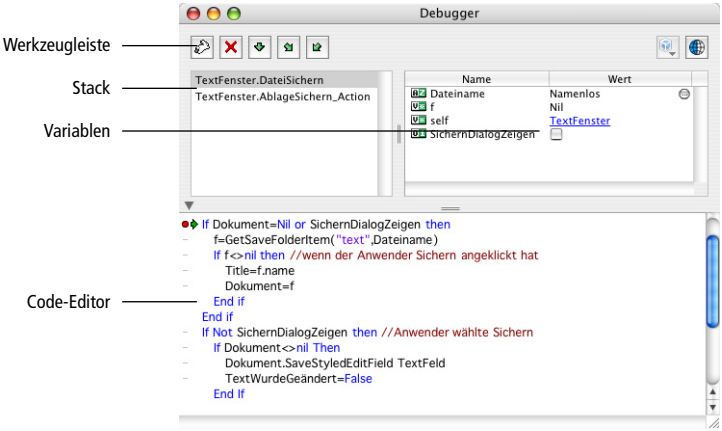
1. Öffnen Sie den Code-Editor für TextFenster.
2. Expandieren Sie im Browser die **Methoden** und selektieren Sie **DateiSichern**.

   Die **DateiSichern**-Methode wird angezeigt. Die Zeilen, in die Sie einen Breakpoint setzen können, sind am Zeilenanfang mit einem Minus („-“) markiert.
3. Klicken Sie in der Zeile, die die erste „if“-Anweisung enthält, auf das Minus. Statt des Minuszeichens erscheint ein roter Punkt – das Symbol für einen Breakpoint:



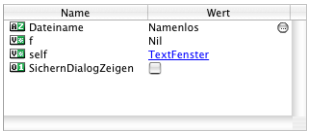
   Dieser Breakpoint wird REALbasic dazu veranlassen, den Programmablauf zu unterbrechen, sobald Sie ein Dokument in der Laufzeitumgebung speichern wollen. Wenn Sie versuchen, ein neues Dokument zu speichern, wird an Stelle der Dateiauswahlbox der Debugger erscheinen.
4. Wählen Sie **Debug/Programm starten**, um das Programm in der Laufzeitumgebung zu starten.
5. Tippen Sie ein wenig Text ein und wählen Sie dann **Ablage/Sichern** (Windows: **Datei/Sichern**) aus.

Ihre Applikation stoppt am Breakpoint und zeigt den Debugger wie unten abgebildet.



Das Debugger-Fenster besteht aus drei Teilen:

- Im *Code-Editor* wird die Methode angezeigt, die gerade ausgeführt wird. In der Abbildung wurde die Programmausführung in der Zeile mit dem Breakpoint (zu erkennen an dem roten Kreis) gestoppt. Der grüne Pfeil markiert die Zeile, die als nächstes ausgeführt wird.
- Der *Stack* zeigt die Aufrufkette, die zum Aufruf der aktuellen Methode geführt hat. Dabei werden die Methoden in der Reihenfolge angezeigt, in der sie aufgerufen wurden (die aktuelle Methode steht oben). Mit Hilfe des Stacks können Sie also überprüfen, ob eine Methode wie erwartet aufgerufen wurde.
- Die *Variablenliste* enthält alle lokalen Variablen der aktuellen Methode samt deren Werte (falls vorhanden). Den Datentyp können Sie am Symbol vor dem Variablennamen erkennen.



Bei der Variablen **SichernDialogZeigen** handelt es sich um den Parameter, der an die Methode übergeben wurde. Diese Variable ist vom Typ Boolean und enthält derzeit den Wert False, was durch eine nicht aktive Checkbox dargestellt wird. Der Wert des String-Parameters ist „Namenlos“.

Objekte, die in der Methode definiert wurden und die bereits einen Wert haben, erscheinen blau unterstrichen als Hyperlinks. Im Beispiel ist f ein FolderItem, dem noch kein Wert zugewiesen wurde.

Die Variable self ist eine Referenz auf das Fenster selbst und vom Typ TextFenster. Diese Variable können Sie in einem Object-Viewer öffnen, indem Sie auf den Hyperlink „TextFenster“ klicken. Der Object-Viewer enthält eine Liste aller Eigenschaften des Objekts. Je nach Objekttyp gibt es verschiedene Object-Viewer.

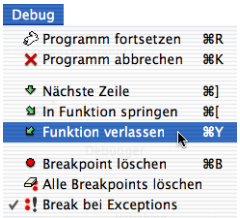
Die Eigenschaften eines Objekts werden im Object-Viewer wie Variablen im Variablenfenster dargestellt. Der Wert einer Eigenschaft, die wiederum ein Objekt ist, verbirgt sich hinter einem Hyperlink auf einen weiteren Object-Viewer. In der Abbildung wird der Wert der Graphics-Eigenschaft als Hyperlink auf ihren Object-Viewer dargestellt. Es können beliebig viele Object-Viewer gleichzeitig geöffnet sein.

Ein Object-Viewer für ein Fenster enthält zwei Karteikarten. Die Karteikarte „Inhalt“ listet alle Steuerelemente des Fensters auf. Da es sich bei diesen wiederum um Objekte handelt, werden sie als Hyperlinks dargestellt:

Variablenliste und Object-Viewer sind interaktiv und werden in Echtzeit aktualisiert, während Sie Ihr Programm im Debugger schrittweise abarbeiten. Damit können Sie verfolgen, ob ein Problem durch einen bestimmten Wert (oder einen fehlenden Wert) verursacht wird.

Die Werkzeugleiste des Debuggers enthält fünf Knöpfe, die einen direkten Zugriff auf Funktionen des **Debug**-Menüs gestatten. Mit den Knöpfen können Sie die Ausführung Ihres Programms steuern. Es ist möglich, den Programmcode Zeile für Zeile auszuführen.

Das **Debug**-Menü enthält die gleichen Funktionen wie die Werkzeugleiste des Debuggers:



- **Programm fortsetzen** (in der IDE: **Programm starten**): Setzt die Ausführung des Codes bis zum nächsten Breakpoint fort. Mit diesem Befehl verlassen Sie das Debugger-Fenster.
- **Programm abbrechen**: Bricht die Ausführung des Programmcodes ab und bringt Sie zurück zur REALbasic Entwicklungsumgebung.
- **Nächste Zeile**: Führt die aktuelle – mit einem grünen Pfeil markierte – Programmzeile aus. Springt zur nächsten Zeile. Wenn die aktuelle Zeile eine Methode beinhaltet, führt der Debugger diese Methode auf einmal aus, arbeitet sie also nicht zeilenweise ab.
- **In Funktion springen**: Führt die aktuelle Zeile aus und springt auf die nächste. Falls die aktuelle Zeile einen Methodenaufruf enthält, zeigt der Debugger deren Code an und erlaubt es, diesen ebenfalls zeilenweise abzuarbeiten.

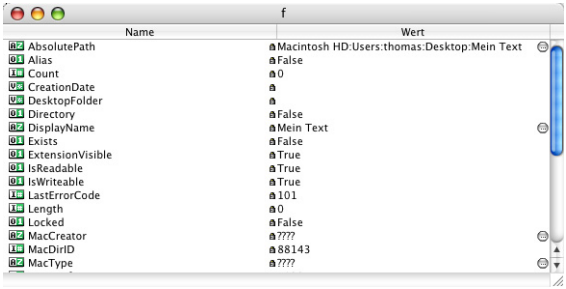
- **Funktion verlassen**: Die aktuelle Methode wird bis zum Ende ausgeführt, ohne dass jede Zeile einzeln bestätigt werden muss. Das ist besonders dann sehr praktisch, wenn Sie vorher mit dem Befehl **In Funktion springen** in die Methode gesprungen sind und diese nun wieder verlassen wollen.

  Im **Variablen**-Fenster sehen Sie, dass die Variable **f** undefiniert ist. So sollte es auch sein, so lange Sie das Dokument noch nicht gespeichert haben. Die Variable **f** wird definiert, wenn Sie das Dokument abspeichern.

  Im Debugger können Sie Zeile für Zeile abarbeiten und den Inhalt des Variablen-Fensters beobachten. Sie erreichen dies durch die Knöpfe **Nächste Zeile** oder **In Funktion springen** (bzw. deren Menü-Pendants).

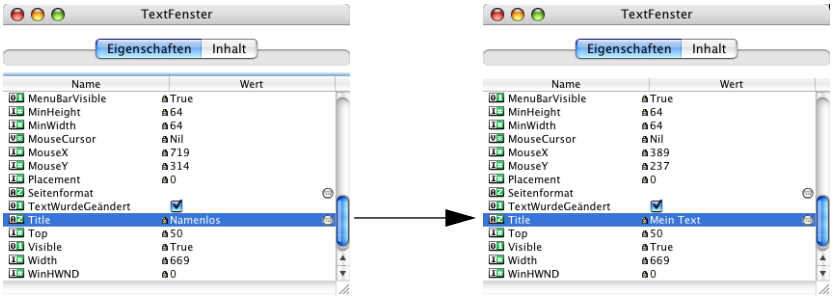
6. Klicken Sie in der Werkzeugleiste so oft auf den Knopf , bis der Sichern-Dialog erscheint. Mit jedem Klick wird eine Programmzeile abgearbeitet.
7. Speichern Sie das Dokument unter einem beliebigen Namen und betrachten Sie dann die Variablenliste.

   Der Wert der Variablen **f** hat sich von Nil auf FolderItem geändert, da er nun definiert ist. Klicken Sie auf den FolderItem-Hyperlink, um **f** im Object-Viewer anzuzeigen:



   Hier sehen Sie u.a. den absoluten Pfadnamen des Dokuments und den Dateinamen, den Sie vergeben haben.
8. Klicken Sie in der Variablenliste auf den Hyperlink TextFenster, um dessen Object-Viewer zu öffnen. Scrollen Sie nach unten, bis Sie die Title-Eigenschaft im Blick haben.

   Klicken Sie auf den Knopf . Sobald die Quelltextzeile `Title=f.name` ausgeführt wurde, ändert sich die Title-Eigenschaft auf den von Ihnen eingegebenen Dateinamen:



9. Klicken Sie erneut auf den Knopf , um die Zeile `Dokument=f` auszuführen.

   Dabei werden Sie feststellen, dass sich die Dokument-Eigenschaft von TextFenster von Nil auf FolderItem ändert.
10. Klicken Sie auf den Knopf , um das Programm fortzusetzen.
11. Beenden Sie Ihr Programm und kehren Sie zur Entwicklungsumgebung zurück.

Laufzeitfehler abfangen

Manche Fehler treten erst auf, wenn eine Anweisung unter bestimmten Bedingungen ausgeführt wird. Diese Fehler werden *Laufzeitfehler* genannt und können nicht bei der Syntaxüberprüfung gefunden werden. Wenn Sie Laufzeitfehler nicht abfangen, stürzt die Stand-Alone-Version Ihrer Applikation ab, sobald die fehlerhafte Anweisung ausgeführt wird.

Das Vorhandensein eines möglichen Laufzeitfehlers hindert REALbasic nicht daran, die Applikation erfolgreich zu compilieren. Die Applikation läuft so lange ohne Probleme, bis der fehlerhafte Code ausgeführt wird. Wenn sich der Fehler z.B. in einer If-Abfrage befindet und die Bedingung niemals True wird, läuft die Applikation ganz normal.

Auf Laufzeitfehler können Sie mit der **RunTimeException**-Klasse reagieren.

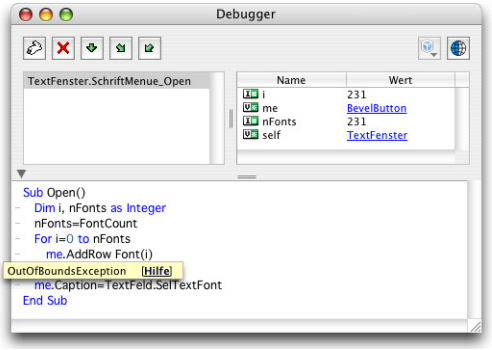
Um zu Testzwecken einen Laufzeitfehler zu erzeugen, können Sie folgendes tun:

1. Stellen Sie sicher, dass im Debug-Menü der Menüpunkt **Break bei Exceptions** mit einem Häkchen versehen ist.
2. Expandieren Sie im Code-Editor den Steuerelemente-Eintrag von TextFenster und öffnen Sie den **Schrift-Menue**-Event-Handler.
3. Selektieren Sie das Open-Event. Sie sehen folgenden Code:

```
Dim i, nFonts as Integer
nFonts=FontCount-1
For i=0 to nFonts
    me.AddRow Font(i)
Next
me.Caption=TextFeld.SelTextFont
```

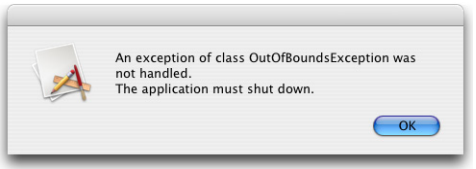
4. Ändern Sie die Definition von **nFonts=FontCount-1** auf **nFonts=FontCount**.
5. Wählen Sie **Debug/Programm starten**.

   Die Ausführung stoppt und Sie erhalten folgende Fehlermeldung:



   Der Fehler tritt auf, sobald **i** in der **For**-Schleife den Wert **FontCount** erreicht hat. Da die Menüeinträge aber mit 0 beginnend durchnummeriert sind, versucht dieser Code, einen Menüeintrag für eine nicht vorhandene Schrift zu erzeugen. Der Wert liegt also außerhalb der Indexgrenzen. Dies ist zwar kein Syntaxfehler, führt aber zu einem Laufzeitfehler.

   Wenn ein Laufzeitfehler in einer Stand-Alone-Applikation auftritt, kann REALbasic natürlich nicht die problematische Quelltextzeile anzeigen, in der der Fehler auftrat. Statt dessen erscheint eine Fehlermeldung:



   Mit Klick auf **OK** wird das Programm beendet.

Es gibt einen Weg, wie man Laufzeitfehler eleganter behandeln kann. Mit einem **Exception Block** können Sie Laufzeitfehler abfangen, wenn diese auftreten. Mehr zu diesem Thema finden Sie im Entwicklerhandbuch.

12. Erzeugen eines Stand-Alone-Programms

In diesem Kapitel werden Sie erfahren, wie Sie aus Ihrem Projekt eine Stand-Alone-Applikation für Mac OS, Windows oder Linux erzeugen.

REALbasic Professional enthält einen Cross-Compiler, der Stand-alone-Applikationen für alle drei Plattformen erzeugen kann. REALbasic Standard erzeugt Stand-Alone-Applikationen für die Plattform, auf der Sie es einsetzen und Demo-Applikationen für die anderen Plattformen. Diese Demoversionen beenden sich nach 5 Minuten selbst.

Los geht's

Starten Sie REALbasic und öffnen Sie das Projekt **TextEditor-Kapitel10**. Falls nötig, verwenden Sie die mitgelieferte Projektdatei aus dem Ordner **Tutorial Projekt**.

Die Dialogbox „Compiler-Einstellungen"

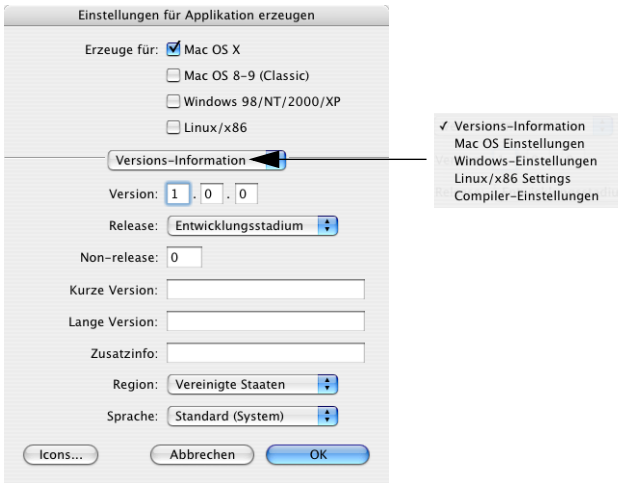
Wenn Sie Ihr Projekt ausgetestet haben und alles wie erwartet funktioniert, wollen Sie es bestimmt in ein eigenständig lauffähiges Programm verwandeln. Als Stand-alone-Programm funktioniert Ihr Programm unabhängig von der REALbasic-Entwicklungsumgebung wie jedes andere Mac OS-, Windows- oder Linux-Programm.

Das Ablage-Menü (Windows: **Datei-Menü**) enthält zwei Einträge, die Sie zum Erzeugen von Stand-alone-Applikationen benötigen: **Compiler-Einstellungen** und **Applikation erzeugen**. Der Menüpunkt **Compiler-Einstellungen** zeigt eine Dialogbox an, in der Sie die Parameter für das Erzeugen der Applikation festlegen. Der Menüpunkt **Applikation erzeugen** erzeugt schließlich die Stand-alone-Applikation und verwendet hierzu die Parameter, die Sie im Dialog **Compiler-Einstellungen** vorgegeben haben.

Aus diesem Grund öffnen Sie normalerweise zuerst den Dialog **Compiler-Einstellungen** und erzeugen danach durch **Applikation erzeugen** Ihre Applikation.

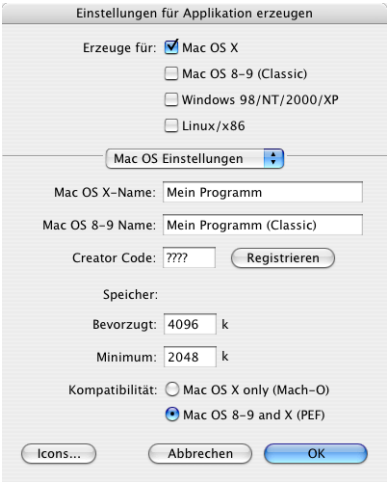
So erzeugen Sie aus einem Projekt ein Stand-alone-Programm:

1. Wählen Sie **Ablage/Compiler-Einstellungen…** bzw. **Datei/Compiler-Einstellungen** unter Windows.



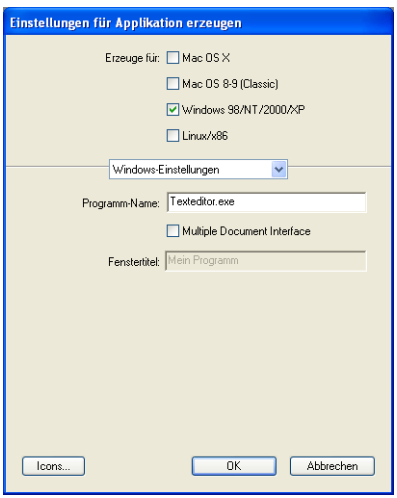
Im oberen Bereich der Dialogbox wählen Sie die Zielplattform, für die Sie Ihr Projekt compilieren wollen. Sie haben die Wahl zwischen Mac OS X, Mac OS 8/9, Windows 98/ME/NT/2000/XP und Linux für x86-Computer mit installiertem GTK+ in Version 2 oder höher. Wenn Sie für mehrere Plattformen gleichzeitig compilieren wollen, setzen Sie einfach die entsprechenden Häkchen.

2. Als Vorgabe ist die Plattform ausgewählt, auf der Sie arbeiten. Wenn Ihnen auch andere Betriebssysteme zum Testen zur Verfügung stehen, selektieren Sie diese zusätzlich.
3. Wenn Sie unter Mac OS arbeiten, wählen Sie **Mac OS Einstellungen** aus dem Popup-Menü:



Geben Sie – je nachdem, welches Betriebssystem Sie verwenden, entweder als Mac OS X-Name oder als Mac OS 8-9-Name **TextEditor** ein.

4. Wenn Sie unter Windows arbeiten, wählen Sie **Windows-Einstellungen** aus dem Popup-Menü:



Geben Sie **TextEditor.exe** unter **Programm-Name** ein.

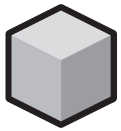
5. Wenn Sie für Linux compilieren, wählen Sie Linux/x86-Einstellungen aus dem Popup-Menü und tragen als Programm-Name **Texteditor** ein.
6. Klicken Sie auf **OK**, um die Einstellungen zu übernehmen und den Dialog zu schließen.

7. Rufen Sie danach den Menüpunkt **Ablage/Applikation erzeugen** (Windows: **Datei/Applikation erzeugen**) auf.

   REALbasic erzeugt für alle im Compiler-Einstellungen-Dialog gewählten Plattformen Stand-alone-Programme, legt diese im Projektordner ab und öffnet diesen Ordner auf dem Schreibtisch.

   Sie können nun REALbasic beenden, einen Doppelklick auf das **TextEditor**-Symbol ausführen und nach Herzenslust Text erfassen.

   Wenn Sie für weitere Plattformen Stand-alone-Programme erzeugt haben, können Sie diese nun ebenfalls unter diesen Plattformen testen.



REALbasic®

ENTWICKLERHANDBUCH

Kapitel 1

Einführung

Inhalt

Willkommen bei REALbasic

Mit REALbasic können einfache und komplexe Programme sehr schnell entwickelt werden. Einsteiger werden feststellen, dass REALbasic einfach zu erlernen ist. Erfahrene Entwickler werden zu schätzen wissen, wie schnell man mit REALbasic das gewünschte Ziel erreicht.

Die grafische Benutzerschnittstelle, über die Ihr Programm mit dem Benutzer kommuniziert, nennt man Graphical User Interface oder auch GUI. Die Benutzerschnittstelle wird in REALbasic mit dem Graphical User Interface Builder zusammengesetzt. Wenn Sie wissen, wie man mit der Maus per Drag & Drop Elemente verschiebt, dann sind Sie in der Lage, mit dem GUI-Builder eine Benutzerschnittstelle aufzubauen. REALbasic bietet eine große Auswahl fertiger Steuerelemente und gestattet dem Anwender darüber hinaus, eigene Steuerelemente zu entwerfen.

Die Entwicklungsumgebung wird auch als Integrated Development Environment (kurz IDE) bezeichnet, da sie alle Komponenten zusammenfasst, die im Laufe der Entwicklung benötigt werden.

Die in REALbasic verwendete Programmiersprache ist eine objektorientierte Version von BASIC. BASIC steht für Beginners All-Purpose Symbolic Instruction Code (frei übersetzt: universelle Programmiersprache für Anfänger). Ursprünglich wurde BASIC für Lehrzwecke entwickelt. Daher ist die Syntax von BASIC auch weniger kompliziert als die der meisten anderen Programmiersprachen. REALbasic kennt die meisten Befehle, die es auch in BASIC gibt. Allerdings hören damit die Ähnlichkeiten zwischen BASIC und REALbasic auch schon auf.

Die meisten BASIC-Dialekte werden interpretiert. Das bedeutet, dass in der Entwicklungsumgebung ein Übersetzer (Interpreter) eingebaut ist, der ständig damit beschäftigt ist, jeden BASIC-Befehl in die tatsächliche Sprache des Prozessors (Maschinencode) zu übersetzen. REALbasic hat keinen Interpreter, sondern verwendet einen Compiler, der den BASIC-Code vorübersetzt, so dass bei der Ausführung des Programms der Hemmschuh der ständigen Übersetzung entfällt. Tatsächlich hat REALbasic sogar einen dynamischen Recompiler. Bei jedem Start des Projekts werden also immer nur die geänderten Teile compiliert und nicht das gesamte Projekt, was die Geschwindigkeit erheblich steigert.

Die in REALbasic verwendete BASIC-Variante ist objektorientiert. Sie stützt sich auf eine moderne Struktur, wie sie auch bei C++ oder Java zum Einsatz kommt. Das objektorientierte Modell teilt ein Programm in einzelne Objekte auf, deren Verhalten man definiert, so wie man es auch für Gegenstände in der wirklichen Welt tun könnte. Dies erleichtert den Überblick und die Fehlersuche. Dabei ist REALbasic in vielerlei Hinsicht erheblich konsequenter als beispielsweise C++. Und auch einfacher zu erlernen.

Da REALbasic es Ihnen erspart, die Programmierung der Benutzerschnittstelle des Betriebssystems zu erlernen, geht die Entwicklung sehr viel schneller als mit anderen Programmiersprachen von der Hand. Das Application Programming Interface (API) des Mac OS umfasst viele tausend Kommandos (Toolbox-Funktionen), die Sie nicht kennen müssen, wenn Sie mit REALbasic programmieren.

REALbasic installieren

Für REALbasic auf dem Macintosh benötigen Sie folgende Systemvoraussetzungen:

- Einen Macintosh mit PowerPC-Prozessor.
- Mac OS Classic 8.5-9.2.x. Mac OS 8.5 ist die empfohlene Minimalvoraussetzung für Mac OS Classic.
- Falls Sie Mac OS X verwenden, ist Version 10.1 die empfohlene Minimalvoraussetzung.
- Für Mac OS 8-9 benötigen Sie mindestens 10 MB verfügbaren Speicher und der virtuelle Speicher sollte aktiviert sein. Weisen Sie dem Programm 2 MB mehr zu, wenn virtueller Speicher ausgeschaltet ist. Mac OS X verwaltet den Speicher dynamisch.
- Mindestens 20 MB Speicher sollte auf der Festplatte für die Installation von REALbasic frei sein.

Kopieren Sie das REALbasic-Programm, die Dokumentation und die Beispiele von der CD auf die Festplatte.

Für die Installation unter Windows benötigen Sie folgende Systemvoraussetzungen:

- Einen PC mit mindestens 200 MHz.
- Windows 98, NT, 2000 oder XP.

Verwenden Sie den Windows-Installer von der REALbasic-CD, um die Windows-Version zu installieren.

Womit fängt man an?

Wenn Sie gleich mit dem Programmieren beginnen wollen, sollten Sie sich den REALbasic QuickStart und das Tutorial ansehen. Dort finden Sie einen Überblick über das System und die Programmiersprache. Lesen Sie dann dieses Benutzerhandbuch. Hier finden Sie detaillierte Erläuterungen zur Sprache und den anderen Komponenten, die es in REALbasic gibt. Wenn Sie Informationen zu einem bestimmten Kommando oder Steuerelement suchen, dann schlagen Sie in der Sprachreferenz nach.

Aufbau der Dokumentation

Menüpunkte

Menüpunkte werden wie folgt beschrieben: **Datei/Beenden** (⌘-Q) bedeutet, dass der Menüpunkt **Beenden** im Menü **Datei** gewählt werden soll.

Tastatur-Shortcuts

Die allermeisten Menüpunkte besitzen auch einen Tastaturbefehl. Auf dem Macintosh ist es die Befehlstaste (⌘), die am häufigsten zur Eingabe von Tastaturbefehlen eingesetzt wird. Auswahl (alt)- und Shift-Taste kommen ebenfalls vor. Unter Windows werden die Strg-Taste und manchmal auch die Alt- und Shift-Tasten benutzt. In der folgenden Anleitung wird immer zuerst der Macintosh-, dann der Windows-Tastaturbefehl angegeben. So bedeutet „⌘-Q oder Strg-Q“ Befehlstaste-Q auf dem Macintosh und Strg-Q unter Windows.

Bildschirmfotos

REALbasic ist eine echte Cross-Plattform-Anwendung. Das Programm läuft unter Mac OS Classic (ab Version 8.5), Mac OS X und Windows. Alle mit REALbasic erzeugten Anwendungen laufen sowohl auf beiden Macintosh Betriebssystemen als auch auf Windows-Betriebssystemen, von Windows 98 bis Windows XP Professional. Außerdem werden diverse Linux-Distributionen für die x86-Plattform unterstützt. Die Bildschirmfotos in diesem Handbuch sind eine Mischung aus

Mac OS Classic, Mac OS X, Windows XP und Linux (GTK 2.x). Wo es notwendig ist, werden die Fenster und Steuerelemente von fertigen Applikationen für alle vier Plattformen gezeigt.

Code-Beispiele

Code-Beispiele werden immer wie folgt abgebildet:

```
Dim i, x as Integer
For i = 1 to 100
       x = x + i
Next
```

Wir haben meist darauf verzichtet, die Codebeispiele ins Deutsche zu übersetzen. Erfahrungsgemäß ist dies nur eine weitere Fehlerquelle, die das Verständnis eher erschwert als erleichtert.

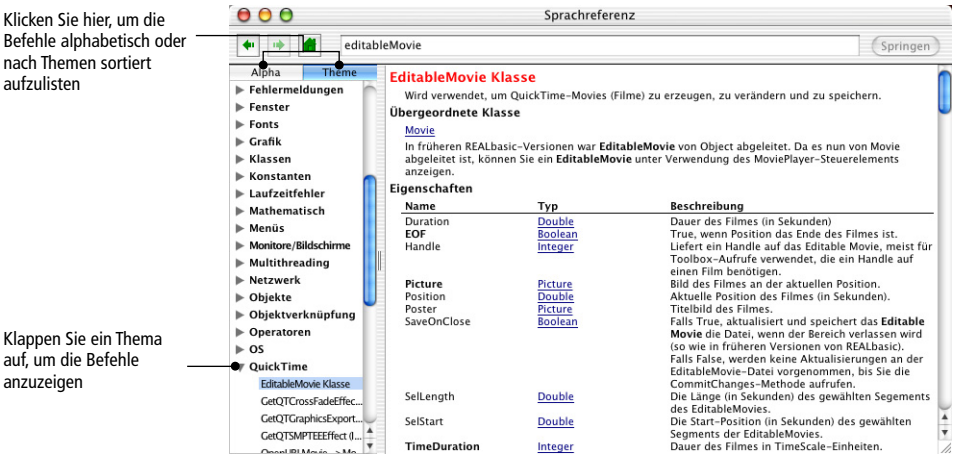
Macintosh Betriebssysteme

Mit Mac OS X verwenden Macintosh-Computer jetzt ein modernes, auf Unix basierendes Betriebssystem, das mit Funktionen wie präemptivem Multitasking, dynamischer Speicherverwaltung, eingebauter Multiprozessorunterstützung und mit einem neuen Interface aufwartet. In der REALbasic Dokumentation werden Mac OS-Versionen vor Mac OS X unter dem Begriff Mac OS Classic zusammengefasst.

Benutzen der Online-Referenz

Eine elektronische Version der REALbasic-Sprachreferenz ist in REALbasic eingebaut. Sie ist unter **Hilfe/Sprachreferenz** zu finden (⌘-1 oder F1 unter Windows).

Abb. 1: Die Online-Referenz



Alle Haupteinträge der gedruckten Referenz werden im Browser im linken Teil des Fensters aufgelistet.

Sie können zwischen alphabetischer oder thematischer Anzeige wählen. Klicken Sie hierzu auf einen der Knöpfe **Alpha** oder **Theme**. Wenn Sie mit der thematischen Liste arbeiten, können Sie die Sprachelemente eines Themas mit den Dreiecken aus- und einklappen.

Sie können einen Eintrag durch Anklicken des Namens im Browser anzeigen. Mit den Pfeilknöpfen im Kopfbereich können Sie durch die bereits abgerufenen Einträge blättern.

Die Online-Hilfe durchsuchen

Über die Eingabezeile im Kopfbereich des Fensters können Sie die Online-Hilfe durchsuchen.

Um nach einem REALbasic-Objekt zu suchen, müssen Sie nur den Anfang seines Namens tippen. Dabei versucht REALbasic, den vollständigen Namen des Objekts zu erraten. Wenn es eine Vermutung hat, erscheint diese in grauer Schrift. Um diesen Vorschlag anzunehmen, können Sie die Tab-Taste drücken. Wenn es verschiedene Vorschläge gibt, erscheinen drei Punkte. Nach Drücken der Tab-Taste erscheint ein Popup-Menü mit den Vorschlägen. Mit den Cursortasten können Sie den gewünschten Eintrag ansteuern und dann Tab oder Return drücken, um ihn auszuwählen.

Hinweis: Die Suchfunktion unterstützt keine Mehrwortsuche, Sie können im Suchstring keine Leerzeichen verwenden.

Klicken Sie auf **Suchen** oder drücken Sie die Return-Taste, um die Suche zu starten. Die Online-Hilfe zeigt die Seite im Text-Panel an. Ist der gesuchte Begriff im Browser sichtbar, wird er dort hervorgehoben. Sind im Browser nur die Themen eingeblendet, wird das Thema, das den gesuchten Begriff enthält, hervorgehoben.

Sie können auch nach einem Begriff suchen, für den es keinen eigenen Eintrag in der Sprachreferenz gibt. Ist REALbasic bei der Suche fündig, werden im Browserbereich alle Seiten aufgelistet, die den gesuchten Begriff enthalten.

Wenn REALbasic den Ausdruck nicht finden konnte, erscheint „Suche fehlgeschlagen“ im Browser-Panel. Der Knopf mit dem Häuschen im Kopfbereich bringt das Browser-Panel wieder in den Ursprungszustand zurück und zeigt im Hauptpanel die Einführung der Sprachreferenz an.

In den **Einstellungen** können Sie den Suchen-Knopf in einen Springen-Knopf verwandeln. Wenn Sie auf **Springen** klicken, versucht die Online-Hilfe, genau das Objekt, nach dem Sie suchen, zu finden. Wenn Sie nicht nach einem Sprachobjekt suchen, halten Sie die alt-Taste gedrückt und wechseln zurück auf **Suchen**. Die Online-Hilfe versucht dann, alle Sprachobjekte zu finden, die den gesuchten Ausdruck enthalten.

Sie können den Browser auf die Standardansicht zurückschalten, indem Sie entweder auf **Alpha** oder **Theme** klicken.

Kontextsensitive Hilfe

Die Online-Referenz ist kontextsensitiv. Wenn Sie also ein Steuerelement in der Entwicklungsumgebung selektieren und dann die Online-Referenz öffnen, erhalten Sie Informationen zum selektierten Steuerelement. Ebenso können Sie einen Befehl im Code-Editor selektieren und dann die Online-Referenz öffnen, um Hilfe zu diesem Befehl zu erhalten.

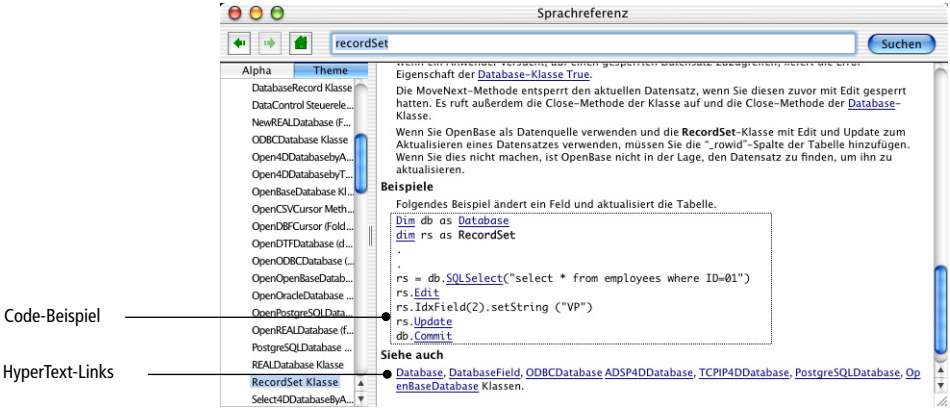
Kontextsensitive Fehlermeldungen

Beim Compilieren eines Programms kann REALbasic nur die korrekte Syntax des Quelltextes überprüfen, logische Fehler treten aber erst auf, wenn das Programm gestartet wird. Es kann also passieren, dass REALbasic auf einen Laufzeitfehler stößt. Dann erscheint im Code-Editor ein Tippsfenster mit einer kurzen Beschreibung des Fehlers. Das Tippsfenster enthält einen Hilfe-Knopf. Wenn Sie diesen drücken, öffnet REALbasic die entsprechende Seite der Online-Hilfe mit einer detaillierten Fehlerbeschreibung.

HyperText-Links

HyperText-Links erscheinen in blauer Schrift und unterstrichen. Wenn Sie auf einen solchen Link klicken, verzweigt die Online-Referenz auf die Seite, auf der dieser Begriff erläutert wird.

Abb. 2: HyperText-Link und Code-Beispiel in der Online-Referenz:



Code-Beispiele

Die Online-Referenz enthält eine Menge Code-Beispiele. Diese sind an dem Rechteck, das sie umgibt, zu erkennen. Sie können solche Code-Beispiele mit der Maus aus dem Referenzfenster in das Editorfenster ziehen, um den Beispielcode in Ihrem Programm zu verwenden.

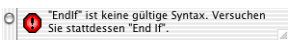
Bei manchen Code-Beispielen wurde aus Gründen der Übersichtlichkeit ein Teil des Codes weggelassen. Dieser wird durch Punkte dargestellt. Hier können Sie Ihren eigenen Code einfügen. Des weiteren enthalten einige Beispiele Unterprogramme oder Funktionen, die automatisch vom Code-Editor hinzugefügt werden. Wenn Sie ein solches Beispiel in den Code-Editor draggen, müssen Sie die doppelten Anweisungen entfernen.

Tipps anwenden

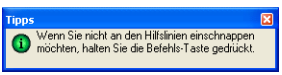
Während Sie arbeiten, „beobachtet" die IDE Ihre Aktivitäten und zeigt Hinweise in einem Tippsfenster an. Wenn Sie zum Beispiel Quelltext eingeben, erinnert Sie das Fenster an die korrekte Syntax der Methode, die Sie gerade eingeben.

Abb. 3: Beispiele für Tipps

Fehlermeldungen



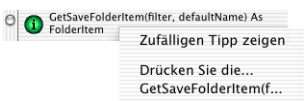
Hilfe-Tipps



Syntax für Methoden

Über das Kontextmenü des Tippfensters können Sie bis zu 5 der vorangegangenen Tipps abrufen oder einen zufälligen Tipp anzeigen. Um in das Kontextmenü zu gelangen, halten Sie über dem Textbereich des Tippfensters die Befehlstaste gedrückt und drücken dabei die Maustaste (unter Windows die rechte Maustaste).

Abb. 4: Das Kontextmenü des Tippfensters



Sie können bestimmte Tipps im Einstellungen-Dialog deaktivieren. Wählen Sie dazu den Menüpunkt **Bearbeiten/Einstellungen** (bzw. unter Mac OS X **REALbasic/Einstellungen**). Wechseln Sie auf die Dialogseite **Tippfenster** und deaktivieren Sie die nicht erwünschten Tipps.

Abb. 5: Tipps-Einstellungen

Sie können auch alle Tipps verstecken (oder sich anzeigen lassen), indem Sie den Menüpunkt **Fenster/Tipps ausblenden** bzw. **Fenster/Tipps einblenden** aufrufen.

Abb. 6: Der Menüpunkt Tipps ausblenden/Tipps einblenden

Weitere Informationsquellen

Elektronische Dokumentation

Die gesamte Dokumentation von REALbasic ist auf der CD enthalten oder im Internet zu finden unter:
http://www.realsoftware.de (deutsche Version) bzw.
http://www.realsoftware.com (englische Version).

Die Dokumente liegen als PDF-Dateien vor und können somit einfach durchsucht werden. Die PDF-Version der Sprachreferenz enthält die gleichen Hyperlinks wie die Online-Hilfe.

Support im Internet

Support im Internet finden Sie unter:
http://www.application-systems.de/support (deutschsprachig) bzw.
http://www.realsoftware.com/support (englisch).

Weiterführende Literatur zu REALbasic

REALbasic Developer (http://www.rbdeveloper.com) ist ein Magazin rund um REALbasic. Hier werden Artikel sowohl von REALbasic-Benutzern als auch von den Entwicklern selbst veröffentlicht.

REALbasic for Dummies (ISBN: 0764507931) von Erick Tejkowski bietet eine Einführung in REALbasic und in die objektorientierte Programmierung für Anfänger. Es ist eine hervorragende Quelle Neueinsteiger.

Ein exzellentes Buch ist *REALbasic: The Definitive Guide, zweite Auflage* von Matt Neuburg (ISBN 0-596-00177-0). Es enthält viele Tipps und Tricks und Programmiertechniken. Sie können das Buch direkt über Application Systems Heidelberg beziehen.

Die REALbasic CD

Wir liefern REALbasic auf einer CD-R aus, damit wir immer gewährleisten können, den aktuellen Stand der Entwicklung herauszugeben. Gegen eine geringe Gebühr können Sie jederzeit eine FreshUp-CD erhalten, die den gerade aktuellen Stand Ihrer REALbasic-Version enthält.

Die Internet Mailing-Lists

Es existieren verschiedene Internet-Mailing-Lists, in denen REALbasic Gegenstand der Diskussion ist. Sie können sich in der englischsprachigen oder auch in der deutschsprachigen Mailing-Liste eintragen. Hier können Sie Fragen stellen oder einfach nur mitlesen und natürlich anderen REALbasic-Benutzern helfen, die Fragen stellen, zu denen Sie Antworten kennen. Sie können sich auf folgenden Web-Seiten zu den Mailing-Lists informieren:
http://www.application-systems.de/realbasic
http://www.realsoftware.com

Application Systems Heidelberg

So können Sie Application Systems Heidelberg kontaktieren:

Application Systems Heidelberg Software GmbH
Pleikartsförsterhof 4/1
69124 Heidelberg, Deutschland
Tel.: 06221-300002, Fax: 06221-300389

E-Mail: support@application-systems.de

Fehler melden und Vorschläge mitteilen

Wenn Sie einen Fehler in REALbasic gefunden haben oder einen Vorschlag zur Erweiterung von REALbasic machen wollen, lassen Sie es uns wissen.

Wenn Sie keinen Internet-Anschluss haben, dann können Sie auch den normalen Postweg oder ein Fax verwenden.

Alpha- und Beta-Versionen

Alle REALbasic-Käufer haben kostenlosen Zugang zu den aktuellen Alpha- und Betaversionen der nächsten REALbasic-Release. Diese stehen (in englischer Sprache) unter http://www.realsoftware.com so lange zur Verfügung, bis die nächste Release veröffentlicht wird.

Kapitel 2 Der Einstieg in REALbasic

Manche Probleme können Sie mit REALbasic binnen weniger Minuten lösen. Zunächst legen Sie das Benutzer-Interface an, das aus Menüs und Fenstern mit Steuerelementen besteht. Danach hauchen Sie mit ein paar Programmzeilen dem Benutzer-Interface Leben ein.

Dieses Kapitel gibt Ihnen einen Überblick über die Entwicklungsumgebung und die Arbeit mit REALbasic-Projekten.

Inhalt

- Das Grundprinzip
- Die Integrierte Entwicklungsumgebung (IDE)
- Die Arbeit mit Projekten

Das Grundprinzip

Einige grundlegende Prinzipien müssen Sie verstehen, um mit REALbasic programmieren zu können.

Events steuern Programme

Bevor Computer grafische Benutzerschnittstellen hatten, wurden Programme als eine Reihe von Befehlen entworfen, die vom ersten Befehl an abgearbeitet wurden, bis eben der letzte Befehl ausgeführt wurde. Die Eingaben des Benutzers wurden nur in Form von Zeichen entgegengenommen. Ein Menü bestand aus durchnumerierten Kommandos, die man über die Tastatur auswählen konnte. Die meiste Zeit warteten die Programme auf eine Benutzerarktion, normalerweise eine Tastatureingabe.

Mit den heutigen grafischen Benutzerschnittstellen hat sich das geändert, denn der Benutzer kann erheblich intuitiver mit dem Programm kommunizieren. Die Programme warten heute immer noch auf Ereignisse, sogenannte Events. Der Unterschied liegt darin, dass es heutzutage jede Menge verschiedene Events gibt, die auftreten können. Früher waren die Programme modal. Das bedeutet, dass es einen Wartemodus gab, in dem das Programm auf ein Event wartete und einen Ausführungsmodus, in dem eine Aktion ablief. Mit einem grafischen Benutzerinterface hat der Benutzer jedoch sehr vielfältige Möglichkeiten, auf ein Programm einzuwirken. Der Benutzer kann relativ frei entscheiden, ob er einen Menüpunkt anwählt, auf einen Knopf klickt oder Daten in ein Feld eintippt. Außerdem kann es sein, dass Programme selbst Events auslösen können, die nur indirekt etwas mit Eingaben des Benutzers zu tun haben, wenn beispielsweise ein Fenster geöffnet oder in der Größe verändert wird.

REALbasic erleichtert es ziemlich, mit allen diesen Events umzugehen. Sie können sehr einfach herausfinden, welche Events in Ihrer Anwendung eine Rolle spielen können. Soll Ihr Programm auf ein Event reagieren, müssen Sie nur das Steuerelement auswählen, das das Event empfangen wird. Dann sehen Sie die möglichen Events, die es empfangen kann und können dazu entsprechende Unterprogramme eingeben. Detailliertere Erläuterungen zu den Events finden Sie weiter hinten in diesem Handbuch.

Programmieren mit REALbasic

Wenn Sie mit anderen Programmiersprachen bereits Erfahrungen haben, dann wissen Sie bereits, dass die Programmentwicklung drei Stufen hat: Code schreiben, compilieren (dabei wird das Programm in etwas übersetzt, das der Computer wirklich versteht) und testen. Gibt es noch Probleme bei der Ausführung, dann geht es wieder von vorne los. Man spricht hier auch von Turnaround und im Zusammenhang mit der Entwicklung ist die Turnaround-Zeit interessant, die

sehr wesentlich davon abhängt, wie lange man beim Compilieren warten muss, denn Änderungen am Programm erfordern immer wieder das Compilieren, um dann im Test wiederum herauszufinden, ob weitere Änderungen erforderlich sind. REALbasic beschleunigt diesen Prozess dadurch, dass der Compiler sehr schnell ist. Deswegen kann man auch nach kleineren Änderungen immer schnell mal testen, ob es so funktioniert, wie man sich das vorstellt, während man bei anderen Entwicklungssystemen immer zunächst sehr viel programmiert, dann einen Compilerlauf startet, der einige Zeit benötigt, um dann viele neue Funktionen zu testen, was den Entwicklungsprozess insbesondere für Anfänger auch komplizierter gestaltet.

Der REALbasic-Compiler weist Sie gleich auf syntaktische Fehler in Ihrem Programm hin. Dabei hebt er direkt im Editor die Stelle im Programm hervor, an der der Fehler aufgetreten ist.

Die Entwicklungsumgebung

REALbasic arbeitet mit einer integrierten Entwicklungsumgebung (*Integrated Development Environment* – kurz IDE). Das heißt, dass alles, was man zum Erzeugen eines Programms benötigt, in einer Applikation zusammengefasst ist: Ein Interface-Builder, ein Code-Editor, ein Compiler und ein Debugger. In traditionellen Programmiersprachen wären diese Teile jeweils ein eigenes Programm. Die IDE von REALbasic besteht aus folgenden Objekten:

Die Menüs

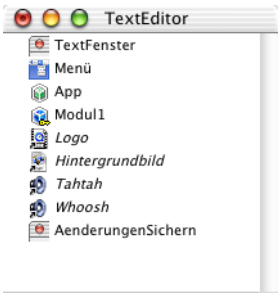
Die Menüleiste bietet Menüs für folgende Aktionen:

- Bearbeiten von Projekten
- Compilieren von Projekten in eigenständige, durch Doppelklick startbare Programme
- Erzeugen neuer Fenster
- Einstellen der Zeichensätze, des Stils und der Größe der Objekte des Benutzerinterfaces
- Positionieren der Objekte im Interface
- Testen und debuggen der Projekte
- Zugriff auf Objekte, die sich im **IDE Extras**-Ordner im REALbasic Ordner befinden
- Auf Informationen der Online-Hilfe und Web-Site zugreifen

Das Projektfenster

Ein Projekt ist die Sammlung aller Elemente, die in einem Programm benötigt werden.

Abb. 7: Das Projektfenster

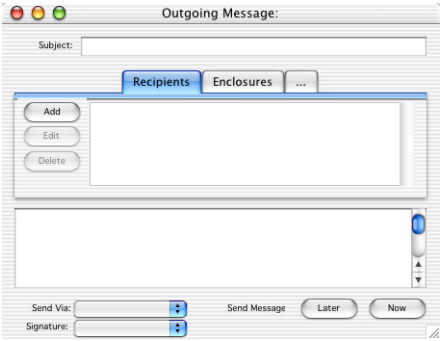


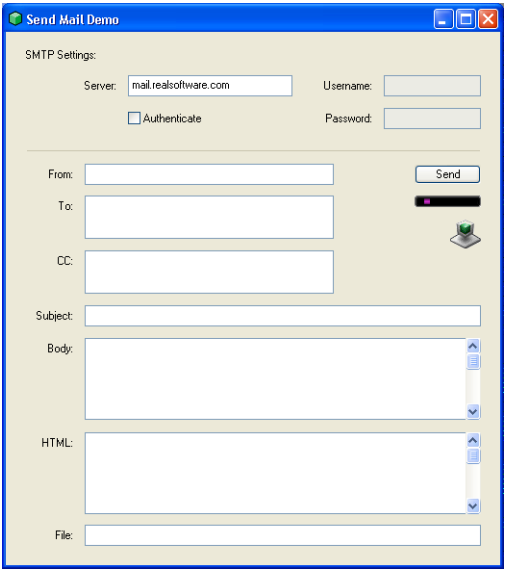
Das Projektfenster zeigt alle diese Elemente. Beispielsweise erscheinen hier alle Fenster Ihres Programms, Bilder, Sound-Dateien, REAL-Datenbanken, QuickTime-Filme, Ressourcen oder andere Elemente, die Sie in Ihrem Programm verwenden. Abbildung 7 zeigt beispielsweise ein Projekt mit einem Quicktime Film und einem Bild. Mehr dazu im nächsten Kapitel.

Der Fenster-Editor

Im Fenster-Editor wird das Aussehen der Benutzerschnittstelle festgelegt.

Abb. 8: Ein Fenster im Fenster-Editor





Mit einem Doppelklick auf ein Fenster, das im Projektfenster aufgeführt ist, öffnet sich der Fenster-Editor. Sie können im Fenster-Editor Ihrem Benutzer-Interface jede Art von Steuerelement hinzufügen und Steuerelemente entfernen oder ändern. Außerdem erreichen Sie hier den zu einem Steuerelement gehörenden Programmcode.

Die Steuerelementepalette

In der Steuerelementepalette finden Sie alle Werkzeuge, mit denen Sie Steuerelemente in Ihr Benutzer-Interface einfügen können und vorhandene Steuerelemente entfernen oder ändern können. Steuerelemente sind Interface-Elemente, wie z.B. Buttons, Checkboxen, Texteingabefelder, Listen, TabPanel, Pop-Up Menüs oder ein Movie-Player. Dazu ziehen Sie einfach das gewünschte Steuerelement aus der Steuerelementepalette auf das Fenster im Fenster-Editor.

Das Eigenschaftenfenster

Eigenschaften (Properties) sind Werte, die zu einem bestimmten Steuerelement gehören. PushButtons haben beispielsweise die Eigenschaft Caption, die den Beschriftungstext für den PushButton enthält. Die Namen der Eigenschaften sind englisch, weil die Programmiersprache sie verwendet und auch die Befehle der Programmiersprache englisch bleiben.

PushButtons haben außerdem die Eigenschaften Left, Top, Width und Height, die Position, Breite und Höhe des Knopfes angeben. Die Eigenschaften, die im Eigenschaftenfenster erscheinen, können hier direkt geändert werden. Sie gelten für das jeweils in der Entwicklungsumgebung selektierte Element. Es Eigenschaften, die man nur per Programmcode verändern kann andere

Eigenschaften, die möglicherweise nur im Eigenschaftenfenster veränderbar sind. Das Aussehen des Eigenschaftenfensters hängt davon ab, welches Objekt selektiert ist. Es kann zur gleichen Zeit immer nur ein einziges Eigenschaftenfenster offen sein.

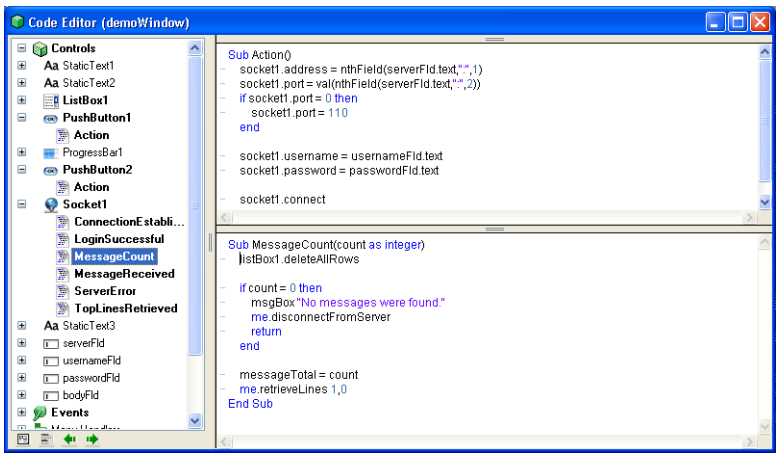
Das Farbenfenster

Farben werden in Werten von 0 bis 255 angegeben. Die 16 Farben, die Sie am häufigsten in Ihrem Projekt verwenden, können Sie im Farbenfenster ablegen. Wenn Sie auf eine Farbe im Farbenfenster klicken, wird der Farbauswahl-Dialog angezeigt, mit dem Sie dem Feld eine neue Farbe zuordnen können. Ein solches „Farbobjekt" können Sie einfach mit der Maus auf die Color-Eigenschaft eines Objekts ziehen, um diesem Objekt die entsprechende Farbe zuzuordnen. Dies gilt sowohl für die Color-Eigenschaft im Eigenschaftenfenster als auch für die Color-Eigenschaft direkt im Code des Code Editor.

Das Code-Editorfenster

In diesem Fenster wird der Programmcode bearbeitet, der Fenstern und Steuerelementen zugeordnet ist. Im Browser sind diese Elemente und ihre Events leicht zugänglich.

Abb. 9: Der Code-Editor (mit zwei Arbeitsbereichen)

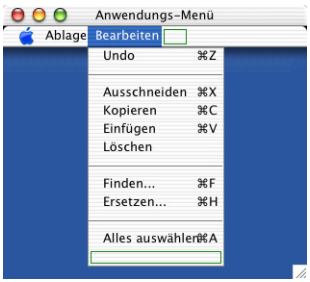


Der Arbeitsbereich des Code-Editors kann in ein oder mehrere Felder unterteilt werden. So können Sie zwei oder mehr Methoden gleichzeitig bearbeiten. Voreingestellt ist ein zusammenhängender Arbeitsbereich.

Der Menü-Editor

Im Menü-Editor werden die Menüs des Programms entworfen.

Abb. 10: Der Menü-Editor

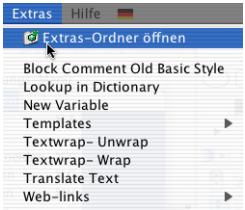


Sie können einzelnen Menüpunkten Tastaturkommandos zuordnen oder auch Untermenüs anlegen (dabei ist dem Menüpunkt dann einfach ein neues Menü zugeordnet). REALbasic legt in der deutschen Version automatisch die Menüs Ablage und Bearbeiten an.

Das Extras Menü

Das **Extras**-Menü verschafft Ihnen unmittelbaren Zugang zum **IDE Extras**-Ordner innerhalb des REALbasic Ordners.

Abb. 11: Das Extras-Menü



Das **Extras**-Menü enthält einen Befehl zum Öffnen des Ordners und Menübefehle für alle Objekte im Wurzelverzeichnis des Ordners. Das macht es zum perfekten Aufbewahrungsort für Ihre Module, Ihre persönlichen Fenster-Vorlagen, Klassen und Codeschnipsel genauso wie für alle Anwendungen, die Sie im Zusammenhang mit REALbasic verwenden. Die Vorlagen, die mit REALbasic geliefert werden, haben die Form von Text-Clips, die in den Code-Editor gedraggt und dort modifiziert werden können. Jedes eigene Objekt (Module, Klassen etc.), das Sie hinzufügen, kann in das Projektfenster gedraggt werden.

Das Hilfe Menü

Über das Hilfe-Menü haben Sie sowohl Zugriff auf die REALbasic Sprachreferenz als auch auf Online-Informationen von der REALbasic-Website. Das Hilfe-Menü besteht aus drei Menüpunkten:

- **Sprachreferenz:** Öffnet die Sprachreferenz.
- **REALbasic Info:** Öffnet den Standard-Webbrowser und geleitet Sie zur Website von REAL Software
- **REALbasic Feedback:** Öffnet die Feedback-Seite von REALbasic. Dort können Sie Fehler melden und Wünsche für neue Funktionen eingeben. Sie können die Feedback-Datenbank nach Stichwörtern durchsuchen.

Die Windows IDE

Die Windows IDE unterscheidet sich von der Macintosh IDE dahingehend, dass sie in einem MDI-Fenster (Multiple Document Interface) angezeigt wird. Code-Editor, Fenster-Editor und Projektfenster befinden sich in dem MDI-Fenster, während die Steuerelementepalette-, das Eigenschaften- und das Farbenfenster schwimmende Fenster sind, die auch aus dem MDI-Fenster gezogen werden können.

Unter Windows haben Sie die Möglichkeit, die Steuerelementepalette und das Eigenschaftenfenster an den Rand des MDI-Fensters anzudocken. Die Steuerelementepalette können Sie auch an die Unterkante des MDI-Fensters andocken.

Um ein Fenster anzudocken, wählen Sie den Menüpunkt **Fenster/Docking** für das jeweilige Fenster und dann die Seite aus, an die das Fenster angedockt werden soll. Dieses Menü wird nur unter Windows angezeigt.

Die Arbeit mit Projekten

In einer Projektdatei sind alle Elemente gespeichert, die zu einem Programm gehören. Dies sind der Programmcode, Plugins, Fenster, die Menüleiste, Klassen, Klassen-Interfaces, Module, Bilder, Sounds, QuickTime-Filme, Datenbanken, AppleEvent-Vorlagen, AppleScript-Skripts, PPC Shared Libraries, Resource-Dateien (nur Mac OS), Cursor-Resources, Ordner und andere Arten von Dokumenten, die als Textstrings behandelt werden. Jedes Element besitzt ein Icon, das seinen Typ anzeigt.

Mit einem Doppelklick auf einen Eintrag im Projektfenster wird das Element angezeigt oder ein Viewer geöffnet, der es anzeigt, falls REALbasic selbst keine Funktion dazu hat. Das funktioniert nicht mit Objekten, die gesperrt sind. Ein Objekt (z.B. eine Klasse) kann von ihrem Entwickler gesperrt werden, damit niemand Zugriff auf den Quelltext hat. So ist es möglich, Erweiterungen für REALbasic zu entwickeln und zu verkaufen. Wenn Sie einen Doppelklick auf ein gesperrtes Objekt ausführen, zeigt REALbasic eine Warnbox an, die mitteilt, dass das Objekt geschützt ist.

Anlegen eines neuen Projekts

Wenn Sie den Menüpunkt **Ablage/Neu** (Windows: **Datei/Neu**) aufrufen, wird der Projekt-Dialog angezeigt. Hier können Sie eine Projektvorlage auswählen. Standardmäßig werden zwei Einträge angezeigt:

Desktop-Applikation: Verwenden Sie diese Vorlage für eine Standard-Applikation mit grafischer Benutzeroberfläche (GUI). Projekte, die ein GUI enthalten, besitzen ein Fenster und ein Menü. Wenn in diesem Entwicklerhandbuch von einem REALbasic-Projekt die Rede ist, ist immer eine Desktop-Applikation gemeint.

Kommandozeilen-Applikation: Vorlage für eine REALbasic-Applikation, die in einem Terminalfenster (Mac OS X und Linux) bzw. in der Kommandozeile (Windows) ausgeführt wird. Verwenden Sie diese Vorlage nur, wenn Ihre Applikation kein GUI benötigt, wie zum Beispiel ein Webserver. Für weitere Informationen lesen Sie die Sprachreferenz zum Thema Kommandozeilen-Applikation.

Standardmäßig enthält das Projekt einer Desktop-Applikation drei Objekte: Fenster1, Menüleiste1 (das Objekt, das die Menüleiste und die Menüeinträge des Programms enthält) und eine spezielle Klasse namens App, die zur Steuerung von globalen Programmfunktionen dient.

Wenn Sie ein neues Projekt über eine Projekt-Vorlage erzeugen, enthält das Projektfenster alle zusätzlichen Objekte, die mit dem Projekt gespeichert wurden, als die Projekt-Vorlage erzeugt wurde.

Hinzufügen von Elementen zu Ihrem Projekt

Ein neues Fenster wird über den entsprechenden Menüpunkt angelegt, während ein Bild, Sound, QuickTime-Film oder eine REAL-Datenbank durch Draggen der Datei vom Desktop auf das Projektfenster in das Projekt importiert wird. Wenn REALbasic ein Alias auf ein externes Objekt speichert, wird dieses kursiv im Projektfenster dargestellt.

Externe Projektelemente

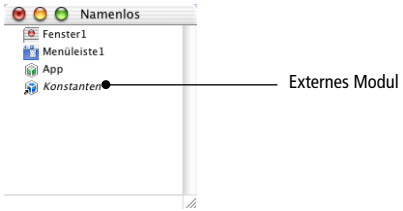
Es ist möglich, Fenster, Klassen oder Module, die in einer externen Datei aufbewahrt werden, in ein Projekt einzubinden. Dieses Feature dient dazu, mehr als einem Projekt den Zugriff auf das extern vorliegende Element zu ermöglichen. Wenn Sie in REALbasic ein extern vorliegendes Element verändern und ihr Projekt speichern, dann werden die von Ihnen vorgenommenen Veränderungen in der externen Datei auf der Festplatte gespeichert. Wenn Sie irgendein anderes Projekt öffnen, das auf die gleiche Datei verweist, wird dieses Projekt die Veränderung übernehmen. REALbasic erlaubt Ihnen jedoch nicht, mehr als eine Kopie von REALbasic zu öffnen, die zeitgleich auf das gleiche externe Element zugreift.

Sie können ein Element, das als externe Datei verwendet werden soll, über das Projektfenster in ein externes Element umwandeln. Klicken Sie dazu mit gleichzeitig gedrückter ctrl-Taste (Windows: rechte Maustaste) auf das Element, rufen das Kontext-Menü auf und wählen dort die Funktion **Externes Element erzeugen** aus. Darauf hin erscheint eine „Datei

sichern"-Dialogbox. Sichern Sie das Element im Format „Standard-Projektbestandteil". Dann wird das Element im Projektfenster durch einen Verweis (Alias) auf das Element ersetzt. Eine Alias-Markierung im Icon des Elements zeigt an, dass das Element nun außerhalb des Projektes liegt und in andere Projekt importiert werden kann.

Wenn Sie ein Element von Ihrem Schreibtisch als externes Element in Ihr Projekt einfügen wollen, halten Sie die Befehls- und die alt-Taste (Strg- und Shift-Taste unter Windows) gedrückt, während Sie das Element von Ihrem Schreibtisch in das Projektfenster ziehen. Das Symbol des Elements im Projektfenster trägt dann eine Alias-Markierung und der Name erscheint in kursiven Buchstaben.

Abb. 12: Externes Modul



Wenn Sie ein Projekt, das externe Elemente enthält, abspeichern, erscheint eine „Sichern als"-Dialogbox für jede externe Datei und anschließend eine für das Projekt selbst. Dies ermöglicht Ihnen, Ihre geänderten Projekt-Elemente an einem neuen Ort abzuspeichern und die Originale unangetastet zu lassen. Externe Projekt-Elemente, die Sie nicht verändert haben, werden in diesem Verfahren nicht berücksichtigt.

Sollte ein externes Projektelement im Finder gesperrt sein, können Sie es in REALbasic nicht verändern, aber trotzdem anzeigen. So werden externe Elemente (die eventuell von mehreren Projekten verwendet werden) vor versehentlichen Veränderungen geschützt. Dadurch besteht auch die Möglichkeit, REALbasic in Verbindung mit Versionskontrollsystemen zu verwenden, die den „checked in/out"-Status mit Hilfe von Dateiattributen abbilden.

Beachten Sie: Wenn eine externe Datei auf der Festplatte von einem anderen Programm als REALbasic geändert wird – wie z.B. einem Versionskontrollsystem – müssen Sie das Projekt neu laden, um auch dieses Element neu zu laden).

Entfernen von Elementen aus Ihrem Projekt

Entfernen kann man Elemente, indem man sie selektiert und die Delete-Taste drückt oder indem man das Objekt selektiert und den Menüpunkt **Bearbeiten/Löschen** aufruft. Sie können dazu auch, wie im folgenden beschrieben, das Kontextmenü verwenden.

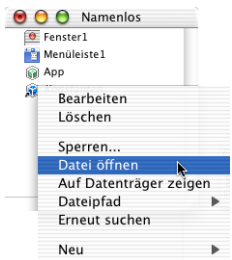
Das Kontextmenü des Projektfensters

Das Projektfenster hat ein eigenes Kontextmenü. Führen Sie einen ctrl-Klick (Macintosh) oder Rechtsklick (Windows) auf ein Projekt-Objekt oder eine freie Stelle des Projektfensters aus, um das Kontextmenü aufzurufen. Die Einträge des Menüs hängen vom Objekttyp ab und können folgende Einträge enthalten:

- **Bearbeiten:** Öffnet das Objekt zum Bearbeiten in REALbasic. Handelt es sich um ein Modul, wird der Code-Editor für das Modul geöffnet. Wenn das Objekt ein Film ist, wird dieser mittels QuickTime geöffnet. Sie können Dateien auch einfach durch einen Doppelklick auf das Objekt im Projektfenster öffnen (wenn das Objekt gesperrt ist, steht der Menüeintrag **Bearbeiten** nicht zur Verfügung).
- **Fenster bearbeiten:** Dieser Menüpunkt steht zur Verfügung, wenn es sich bei dem Objekt um ein Fenster handelt. Er öffnet das Fenster im Fenster-Editor.
- **Code bearbeiten:** Öffnet den Code-Editor für das Fenster. Dieser Menüpunkt wird nur dann angezeigt, wenn es sich bei dem Objekt um ein Fenster handelt.
- **Löschen:** Entfernt das Objekt aus dem Projekt.

- **Ansicht:** Zeigt das Objekt (z.B. ein Bild) in REALbasic an, bietet Ihnen aber nicht die Möglichkeit, es zu bearbeiten.
- **Abspielen:** Dieser Menüpunkt ist nur für eine Sounddatei verfügbar. Spielt den Sound ab.
- **Datei öffnen:** Öffnet die Datei so, als würde sie auf dem Schreibtisch doppelt angeklickt. Wenn das Objekt z.B. ein QuickTime-Film ist, wird es im QuickTime-Player geöffnet.
- **Auf Festplatte anzeigen:** Öffnet den Ordner der Datei und selektiert die Datei.
- **Dateipfad:** Zeigt für das externe Element ein hierarchisches Menü, über das man jeden Ordner innerhalb des Zugriffspfades öffnen kann.
- **Erneut suchen:** Gilt nur für externe Projektelemente. Ruft eine „Datei öffnen"-Dialogbox auf, in der Sie für das Objekt eine neue externe Datei auswählen können.
- **Verschlüsseln:** Ruft die Verschlüsseln-Dialogbox auf und erlaubt Ihnen, ein Passwort einzugeben und das Objekt zu verschlüsseln. Diese Funktion wird normalerweise dazu verwendet, Code von Modulen zu verstecken, die Sie anderen Entwicklern anbieten oder verkaufen. Nur verfügbar, wenn das Objekt nicht bereits verschlüsselt ist. Elemente, die ihren Ursprung außerhalb von REALbasic haben, so wie z.B. Filme, Musik und Bilder, können nicht verschlüsselt werden.
- **Entschlüsseln:** Ruft die Entschlüsseln-Dialogbox auf und fordert Sie auf, das Passwort zum Entschlüsseln einzugeben. Ein verschlüsseltes Objekt wird im Projektfenster mit einem Schloss in der linken unteren Ecke des Icons angezeigt. Nur verfügbar, wenn das Objekt verschlüsselt ist.
- **Neu:** Wird verwendet, um einem Projekt ein neues Objekt hinzuzufügen und hat die gleiche Funktion wie die entsprechenden Menüpunkte aus dem Ablage- (Windows: Datei-) Menü.
- **Neue Unterklasse:** Erscheint nur, wenn das ausgewählte Element im Projektfenster eine Klasse ist. Erstellt eine neue Klasse, fügt sie dem Projektfenster hinzu und passt ihre „Super"-Klasse-Eigenschschaft automatisch der ausgewählten Klasse an. Die neue Klasse enthält automatisch den Namen **BenutzerdefiniertKlasse**, während die „Super"-Klasse-Eigenschaft den eigentlichen Klassennamen trägt. Nachdem Sie die Unterklasse angelegt haben, können Sie mit Hilfe des Eigenschaftenfensters alle nötigen Änderungen an der Unterklasse vornehmen, inklusive ihrer SuperClass-Eigenschaft.
- **Exportieren:** Exportiert das Element auf die Festplatte. Es erscheint eine „Datei sichern"-Dialogbox. Sie können das Element in ein anderes Projekt importieren, indem Sie es vom Schreibtisch auf das Projektfenster ziehen oder es als externes Element importieren. Beim Import als externes Element verbleibt das Element auf der Festplatte und kann in mehreren Projekten gleichzeitig genutzt werden. Dazu müssen Sie die Befehls-(⌘) und Wahltaste(⌥) gedrückt halten (Ctrl/Strg und Shift unter Windows) während Sie das Element vom Schreibtisch auf das Projektfenster ziehen.
- **Externes Element erzeugen:** Zum Abspeichern eines Elements als externe Datei auf der Festplatte. Im Unterschied zu **Exportieren** wandelt **Extern machen** auch das Element des aktuellen Projekts in ein externes Element um. Genauso wie bei exportierten Elementen kann die Kopie auf dem Schreibtisch in andere Projekte importiert werden.
- **Sichern:** Nur für externe Projektelemente. Ruft eine „Datei sichern"-Dialogbox auf und ermöglicht Ihnen, das externe Projektelement auf der Festplatte zu speichern. Dieser Kontextmenüeintrag erscheint nur, wenn Sie an einem externen Projektelement Änderungen vorgenommen haben.

Abb. 13: Das Kontextmenü eines externen Projektelements



Das Projekt speichern

Änderungen am Projekt sichert man über **Ablage/Sichern** (Windows: **Datei/Sichern**). Speichern Sie das Projekt so oft wie möglich. Wenn Sie sich nicht sicher sind, ob Sie die Änderungen, die Sie gerade vornehmen, später behalten möchten, sichern Sie das Projekt mit **Sichern als** unter einem neuen Namen. Dann haben Sie das ursprüngliche Projekt noch in der alten Fassung. Wie bereits erwähnt, können Sie externe Projektelemente unabhängig vom Projekt über deren Kontextmenüs sichern.

Im XML-Format speichern

Projekte können auch im XML-Format gespeichert werden. Rufen Sie dazu den Menüpunkt **Sichern als** auf und wählen Sie im **Format**-Popup **XML**. Dann wird das Projekt in eine XML-Datei exportiert. Das ursprüngliche Projekt wird dadurch nicht überschrieben, sondern verbleibt an der ursprünglichen Stelle im ursprünglichen Format. Wenn Sie den XML-Export unter Mac OS 8 oder 9 verwenden wollen, sollten Sie vorher die Speicherzuteilung an REALbasic erhöhen.

XML laden

Als XML-Dateien vorliegende Projekte können über den Menüpunkt **Ablage/Öffnen** (Windows: **Datei/Öffnen**) importiert werden. Im Dateidialog werden zwar sämtliche Textdateien angezeigt, es können aber nur korrekt formatierte XML-Dateien geöffnet werden. Wenn das Einlesen erfolgreich war, wird ein neues Projekt unter dem Namen der XML-Datei mit angehängtem „(Konvertiert)“ erzeugt.

Hinweis: Wenn Sie eine XML-Datei aus einem AppleScript heraus öffnen wollen, können Sie dieselbe Syntax wie bei normalen Projektdateien verwenden, also: „tell application "REALbasic" to open someFile“.

Projekt-Vorlagen

Wenn Sie bestimmte Elemente in mehreren Projekten verwenden wollen, empfiehlt es sich, ein neues Projekt zu erzeugen, die Elemente hinzufügen und dann das Projekt als Vorlage zu speichern. Wenn Sie später auf Basis dieser Vorlage ein neues Projekt beginnen, erzeugt REALbasic automatisch ein neues, unbenanntes Projekt, das die Elemente der Vorlage bereits enthält. Die Vorlage bleibt dabei unverändert.

Am bequemsten gestaltet sich die Arbeit mit Vorlagen, wenn Sie sie im Verzeichnis „Project Templates“ ablegen, das sich im REALbasic-Verzeichnis befindet. Ihre Vorlagen erscheinen dann in der Vorlagenliste, die bei Auswahl des Menüpunkts **Ablage/Neu** (Windows: **Datei/Neu**) angezeigt wird.

Kapitel 3 Das Benutzer-Interface

Das Benutzer-Interface ist gewissermaßen das Aushängeschild Ihres Programms. Wenn es klar strukturiert ist und sich an die Konventionen des Betriebssystems hält, werden die Anwender auch gut mit Ihrem Programm zurecht kommen.

REALbasic bietet einfache Hilfsmittel, um eine Benutzerschnittstelle schnell und effizient zu entwerfen. Der eingebaute Interface Assistant™ hilft Ihnen, ein sauber standardisiertes Interface aufzubauen.

In diesem Kapitel werden Sie alles lernen, was Sie dazu wissen müssen. Sie lernen die Elemente der Benutzerschnittstelle kennen und werden mit einigen Richtlinien vertraut gemacht, an die man sich halten sollte, wenn man Menüs und Fenster entwirft.

Inhalt

- Arbeiten mit Fenstern
- Interaktion durch Steuerelemente
- Menüs hinzufügen
- Richtlinien für Benutzeroberflächen

Arbeiten mit Fenstern

Normalerweise befindet sich der größte Teil des Benutzer-Interface in Fenstern des Anwendungsprogramms, manche Programme benötigen aber auch gar keine Fenster. Sie gestalten das Benutzer-Interface, indem Sie die benötigten Fenster anlegen und dann Steuerelemente wie PushButtons und Checkboxes hinzufügen. Sie können auch eine Bilddatei vom Desktop direkt auf ein Fenster ziehen, das Bild wird dann als Fensterhintergrund verwendet.

Per Voreinstellungen besitzt eine REALbasic Desktop-Applikation ein Fenster, das angezeigt wird, sobald die Applikation läuft. Normalerweise beginnt man damit, das GUI zu entwerfen, indem man Elemente in diesem Fenster anordnet und schreibt dann die benötigten Methoden.

Um einer Applikation weitere Fenster hinzuzufügen, gehen Sie wie folgt vor:

- Erzeugen Sie ein neues Fenster mit dem Menüpunkt **Ablage/Neues Fenster** (Windows: **Datei/Neues Fenster**).
- Legen Sie in der Eigenschaften-Palette den Typ und andere Eigenschaften des Fensters fest.
- Versehen Sie das Fenster mit Steuerelementen, indem Sie diese aus der Steuerelementepalette auf das Fenster ziehen.
- Schreiben Sie – wo notwenig – den benötigten Quelltext.
- Zeigen Sie das Fenster an (siehe „Fenster öffnen“ auf Seite 204).

Im folgenden Abschnitt gehen wir auf die elf Fenstertypen ein, die von REALbasic unterstützt werden. Zusätzlich, zu diesen Fenstertypen, können Sie Dialoge anzeigen, ohne explizit ein Fenster dafür zu erzeugen. Verwenden Sie dafür die MessageDialog Klasse, die auch in diesem Dokument beschrieben ist.

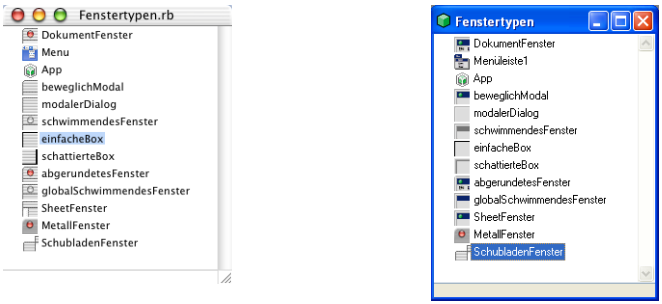
Fenstertypen

REALbasic kennt elf verschiedene Fenstertypen. Einige werden in modernen Anwendungen jedoch nur sehr selten eingesetzt und sind hauptsächlich aus historischen Gründen vorhanden. Andere werden nur von Mac OS X unterstützt. Es folgt eine Liste der Fenstertypen:

- Dokumentfenster
- bewegliche modale Fenster
- modale Dialoge
- schwimmende Fenster
- einfache Boxen
- schattierte Boxen
- abgerundete Fenster (nicht unter Mac OS X)
- globale schwimmende Fenster
- Sheet-Fenster (nur unter Mac OS X)
- Metallfenster (nur unter Mac OS X 10.2 und höher)
- Schubladenfenster (nur unter Mac OS X 10.2 und höher)

Der spätere Einsatzzweck entscheidet darüber, welchen Typ Sie wählen. Jeder Fenstertyp hat sein eigenes Icon im Projektfenster. Dies zeigt die folgende Abbildung:

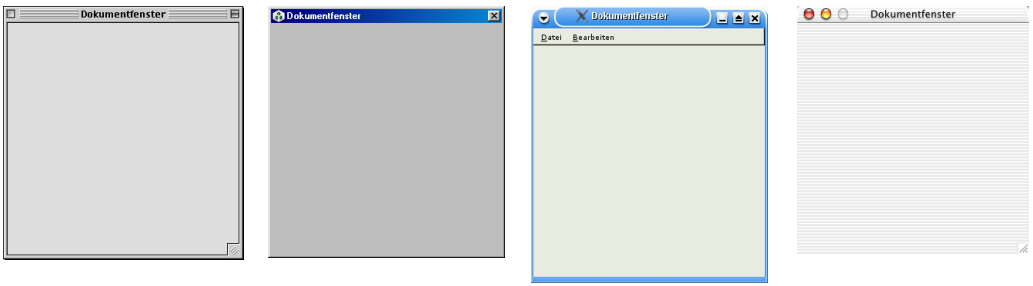
Abb. 14: Projektfenster mit den 11 verschiedenen Standardfenstertypen



Dokumentfenster

Dokumentfenster sind der am häufigsten verwendete Fenstertyp. Der Benutzer kann Dokumentfenster geöffnet lassen und andere Fenster in den Vordergrund holen etc. Dokumentfenster können mit dem Schließknopf (falls vorhanden) oder einem entsprechend programmierten PushButton geschlossen werden.

Abb. 15: Ein Dokumentfenster



Dokumentfenster können einen Schließknopf, einen Zoomknopf und eine Angreifecke (mit der die Fenstergröße verändert werden kann) haben. Unter Mac OS X besitzen die Fenster einen roten, grünen und gelben Knopf in der Titelleiste.

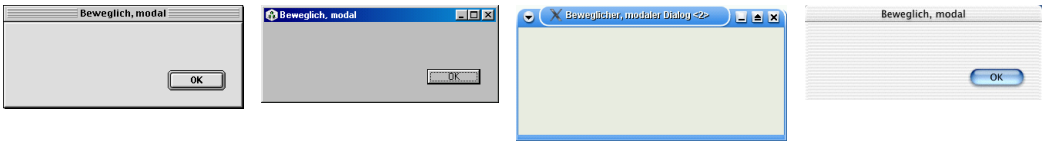
Wenn Sie Ihrem Projekt ein neues Fenster über **Datei/Neues Fenster** hinzufügen, ist es vom Typ Dokumentfenster. Sie können den Typ über die Frame-Eigenschaft in der Eigenschaften-Palette ändern.

Wenn Sie REALbasic zum ersten Mal starten, enthält das voreingestellte Projekt ein leeres Dokumentfenster.

Bewegliche modale Fenster

Dieser Fenstertyp bleibt im Vordergrund und blockiert die Eingabe in andere Fenster des Programms. Man bezeichnet diesen Zustand als modal, da das Programm in einem Modus ist, in dem nur weitergearbeitet werden kann, nachdem das modale Fenster geschlossen wurde. Immer dann, wenn es unbedingt nötig ist, dass der Benutzer eine Entscheidung trifft, ohne die das Programm seine Arbeit nicht sinnvoll fortsetzen kann, muss ein modales Fenster verwendet werden. Da es beweglich ist, kann der Benutzer Informationen in dahinter liegenden Fenstern nach wie vor einsehen.

Abb. 16: Ein bewegliches modales Fenster



Bewegliche modale Fenster können auf dem Macintosh nicht einfach geschlossen werden, da sie eine Eingabe des Benutzers erwarten. Daher verfügen sie nicht über einen Schließknopf. Sie müssen einen entsprechenden Knopf im Fenster vorsehen, es sei denn, das Fenster schließt sich von selbst, nachdem ein bestimmter Vorgang abgeschlossen ist. Solche Fenster auf dem Macintosh können in der Größe nicht verändert werden und haben keine Zoombox. Sie müssen daher die Größe des Fensters selbst festlegen.

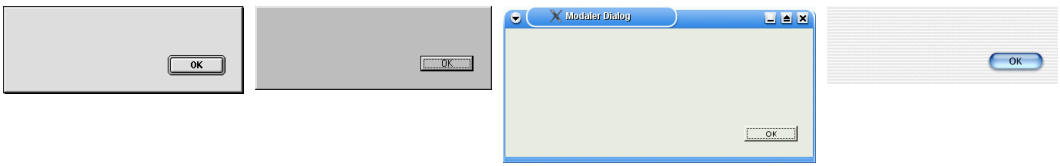
Unter Windows und Linux besitzt ein bewegliches modales Fenster einen Minimieren-, Maximieren- und Schließknopf in der Titelleiste. Bei MDI-Interfaces öffnet sich das Fenster in der Mitte des Bildschirms und nicht in der Mitte des MDI-Fensters. Daher kann es passieren, dass sich das bewegliche modale Fenster außerhalb des MDI-Fensters öffnet.

Hinweis: Es gibt einen Ausnahmefall, in dem ein bewegliches modales Fenster nicht das oberste Fenster ist. Dies ist der Fall, wenn gleichzeitig ein schwimmendes Fenster geöffnet wird. Es ist allerdings schlechter Programmierstil, dies zu tun, denn ein modales Fenster wird ja gerade dann eingesetzt, wenn der Benutzer nichts anderes machen darf.

Modale Dialoge

Diese Fenster sind den beweglichen modalen Fenstern sehr ähnlich. Der einzige Unterschied ist, dass modale Dialoge keine Titelleiste haben, so dass sie nicht bewegt werden können. Unter Windows hat eine modale Dialogbox keinen Minimieren-, Maximieren- oder Schließen-Knopf. In Windows-MDI-Applikationen öffnet sich ein modaler Dialog in der Mitte des Bildschirms und nicht in der Mitte des MDI-Fensters. Daher kann es passieren, dass sich die modale Dialogbox außerhalb des MDI-Fensters öffnet. Unter Linux besitzt ein modaler Dialog eine Titelleiste, einen Schließen- und Minimieren-Button. Die Dialogbox für das Seitenformat ist ein Beispiel für einen solchen modalen Dialog.

Abb. 17: Ein modaler Dialog



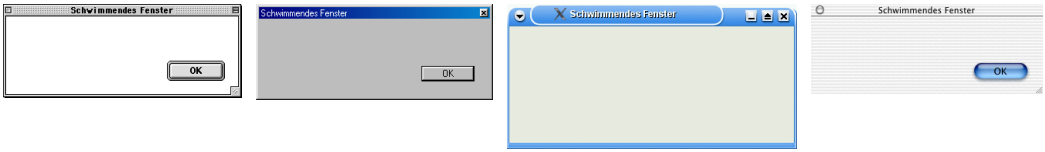
Anmerkung: Die für bewegliche modale Fenster aufgeführte Ausnahme gilt auch für modale Dialoge.

Schwimmende Fenster

Ein schwimmendes Fenster (floating Window oder auch Windoid) schwimmt oben, ähnlich wie die modalen Fenster. Im Unterschied zu diesen bleiben andere Fenster voll bedienbar, auch wenn sie im Hintergrund sind. Wenn Sie mehrere schwimmende Fenster haben, dann können Sie eines dieser Fenster in den Vordergrund holen, aber alle nicht-schwimmenden Fenster bleiben im Hintergrund. Da die schwimmenden Fenster immer oben liegen, sollten sie möglichst klein sein, da sie dem Benutzer sonst im Weg sind. Daher werden diese Fenster meistens für Werkzeugfenster etc. verwendet.

Ein globales schwimmendes Fenster ist ein schwimmendes Fenster, das in der obersten Ebene einer bestimmten Applikation oder in der obersten Ebene der Fenster aller laufenden Applikationen liegt.

Abb. 18: Ein schwimmendes Fenster



Wie Dokumentfenster können schwimmende Fenster einen Schließknopf haben. Sie können auch vom Benutzer in der Größe verändert werden, sie haben jedoch nie eine Zoombox.

In Windows-MDI-Applikationen kann ein schwimmendes Fenster aus dem Fenster der Applikation hinausschwimmen. Normalerweise öffnet sich ein schwimmendes Fenster in einer Windows-MDI-Applikation in der linken oberen Ecke des Bildschirms, egal wo sich das MDI-Fenster befindet.

Einfache Box

Diese Fenster funktionieren wie modale Dialogfenster. Der einzige Unterschied liegt in ihrem Aussehen. Solche Fenster werden nur dann benötigt, wenn eine Anwendung den Desktop überdecken soll. Außerdem werden sie für About-Boxen verwendet.

Abb. 19: Eine einfache Box

In Windows-MDI-Applikationen öffnen sich einfache Boxen in der Mitte des Bildschirms und nicht in der Mitte des MDI-Fensters. Daher kann es passieren, dass sich die einfache Box außerhalb des MDI-Fensters öffnet.

Schattierte Box

Diese Fenster funktionieren wie eine einfache Box. Der einzige Unterschied liegt in ihrem Aussehen. Solche Fenster werden meist für About-Boxen verwendet.

Abb. 20: Schattierte Box

In Windows-MDI-Applikationen öffnen sich schattierte Boxen in der Mitte des Bildschirms und nicht in der Mitte des MDI-Fensters. Daher kann es passieren, dass sich die schattierte Box außerhalb des MDI-Fensters öffnet.

Unter Mac OS X funktioniert eine schattierte Box wie ein modaler Dialog mit einem Minimieren-Knopf.

Abgerundete Fenster

Abgerundete Fenster sind Dokumentfenster, die etwas anders aussehen. Sie können auf dem Macintosh nicht in der Größe verändert werden und haben auch keine Zoombox. Unter Windows hat ein abgerundetes Fenster die Standard-Knöpfe für Minimieren, Maximieren und Schließen in der Titelleiste und die Ecken sind nicht abgerundet. Sie wer-

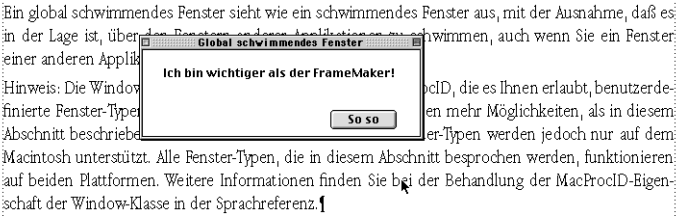
den eigentlich nicht mehr verwendet und es gibt auch keinen Grund, sie einem Dokumentfenster vorzuziehen. Sie sind mehr oder weniger ein Überbleibsel aus alten Tagen des Mac OS. Abgerundete Fenster werden in Mac OS X Applikationen als Dokumentfenster angezeigt.

Abb. 21: Ein abgerundetes Fenster

Global schwimmendes Fenster

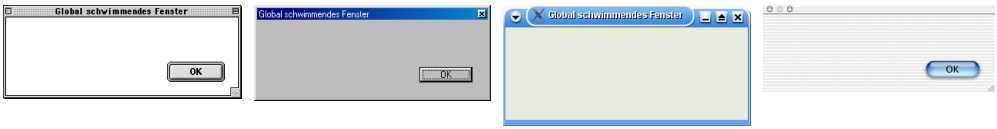
Ein global schwimmendes Fenster sieht wie ein schwimmendes Fenster aus, mit der Ausnahme, dass es in der Lage ist, über den Fenstern anderer Applikationen zu schwimmen, auch wenn Sie ein Fenster einer anderen Applikation nach vorne bringen. In folgender Abbildung wurde auf ein FrameMaker-Fenster mit diesem Text hier geklickt und trotzdem befindet sich das globale schwimmende Fenster darüber.

Abb. 22: Ein globales schwimmendes Fenster über einem Textdokument



Dies funktioniert nicht mit einem schwimmenden Fenster. Ein „normales“ schwimmendes Fenster schwimmt nur über den Fenstern der Applikation, die es geöffnet hat.

Abb. 23: Globale schwimmende Fenster

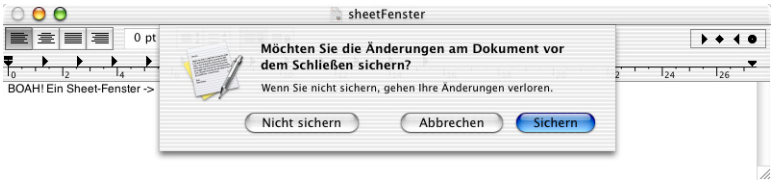


In Windows-MDI-Applikationen kann ein global schwimmendes Fenster aus dem MDI-Fenster hinausschwimmen. Es öffnet sich normalerweise in der linken oberen Ecke des Bildschirms.

Sheet-Fenster

Sheet-Fenster ist der offizielle Name der herabgleitenden Fenster, die mit Mac OS X eingeführt wurden. Sie werden an Stelle der überholten modalen Dialoge verwendet.

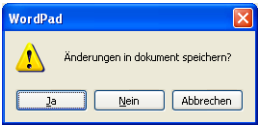
Abb. 24: Sheet-Fenster



Sheet-Fenster verhalten sich genauso wie modale Dialoge. Der einzige Unterschied besteht darin, dass per Animation der Eindruck erweckt wird, als ob das Sheet-Fenster der Titelleiste des Ursprungsfensters entspringt. Es kann nicht verschoben werden und versetzt die Benutzeroberfläche in einen modalen Zustand. Der Benutzer ist gezwungen, auf die Frage im Sheet-Fenster zu reagieren.

Sheet-Fenster verhalten sich nur unter Mac OS X wie Sheet-Fenster. Unter Windows und Linux verhalten sich Sheet-Fenster wie bewegliche modale Dialogfenster und unter Mac OS Classic wie modale Dialoge.

Abb. 25: Sheet-Fenster unter Windows XP



Metallfenster

Ein Metallfenster verwendet den Metall-Look des Mac OS X 10.3-Finders und der iApps von Apple. Ein Metallfenster hat eine Angreifecke zur Größenänderung und die Standard-Titelleiste eines Dokumentfensters mit Knöpfen zum Schließen, Minimieren und Maximieren. Metallfenster tauchen nur unter Mac OS X 1.2 und höher auf.

Abb. 26: Metallfenster

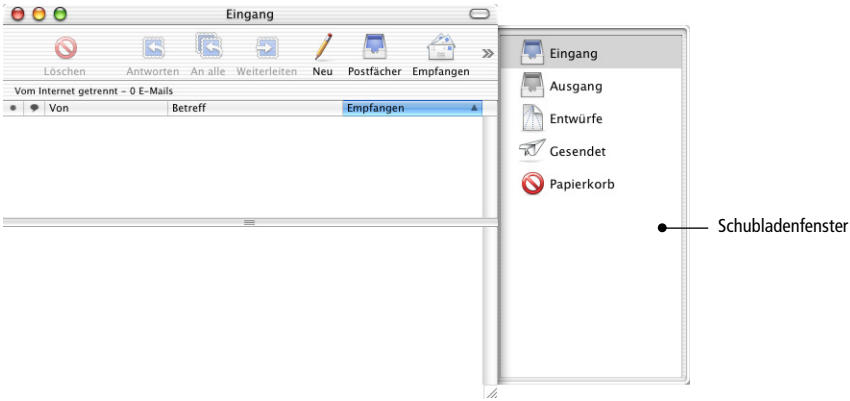
Unter Windows und Mac OS 8/9 wird ein Metallfenster als normales Dokumentfenster dargestellt.

Schubladenfenster

Schubladenfenster wurden auf dem Mac mit Mac OS X 10.2 eingeführt. Sie können oben, unten oder seitlich eine Schublade ausfahren, in der zusätzliche Informationen angezeigt werden. Apple Mail beispielsweise verwendet ein Schubladenfenster zur Anzeige der Mailboxen.

Unter Windows wird ein Schubladenfenster als separates schwimmendes Fenster dargestellt.

Abb. 27: Schubladenfenster bei Mail



Um ein Schubladenfenster anzuzeigen, verwenden Sie die ShowWithin-Methode der Window-Klasse. Damit definieren Sie das Elternfenster, zu dem das Schubladenfenster gehört und legen anschließend fest, aus welcher Seite das Fenster herausfahren soll (links, rechts, oben, unten oder gemäß der System-Voreinstellungen. Bei der letzten Option entscheidet das Betriebssystem, wo sich das Schubladenfenster am besten ausfahren lässt.)

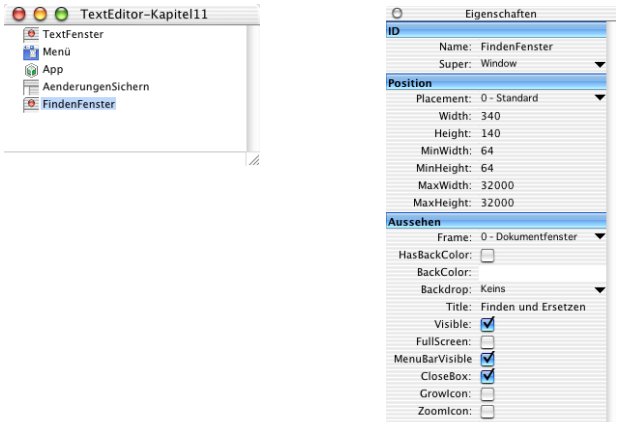
Benutzerdefinierte Fenstertypen

Die Window-Klasse hat eine Eigenschaft namens MacProcID, mit der benutzerdefinierte Fenster-Typen erzeugt werden können. Diese Eigenschaft bietet Ihnen mehr Möglichkeiten, als die in diesem Abschnitt beschriebenen. Benutzerdefinierte Fenster-Typen werden jedoch nur auf dem Macintosh unterstützt. Alle Fenster-Typen, die in diesem Abschnitt besprochen werden, funktionieren auf allen Plattformen mit den im jeweiligen Abschnitt erwähnten Ausnahmen. Weitere Informationen finden Sie bei der Behandlung der MacProcID-Eigenschaft der Window-Klasse in der Sprachreferenz.

Die Fenstereigenschaften eines Fensters verwenden

Wenn Sie im Projektfenster auf den Namen des Fenster klicken, stellt das Eigenschaftenfenster die Eigenschaften dieses Fensters dar, die in der Entwicklungsumgebung gesetzt werden können (im Gegensatz zu den Eigenschaften, die im Code gesetzt werden können). Folgende Abbildung zeigt ein im Projektfenster ausgewähltes Fenster und seine Eigenschaften.

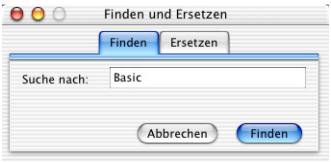
Abb. 28: Ein beweglicher modaler Dialog im Projektfenster und seine Eigenschaften



Sie können die Eigenschaften eines Fensters ändern, indem Sie Werte in die Bereiche im Eigenschaftenfenster eintragen, in denen Eingaben möglich sind, Menüpunkte auswählen oder Checkboxen an- oder ausklicken.

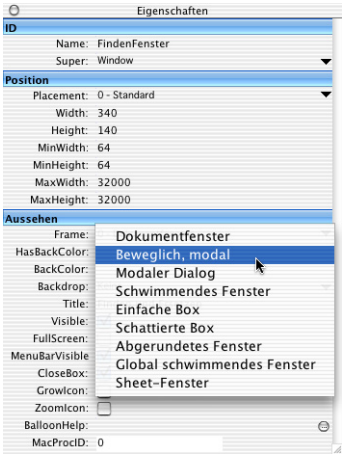
Wenn Sie in ein Eingabefeld Werte eingeben, übernehmen Sie den Wert mit der Return-Taste. In folgendem Beispiel wurde der Fenstertitel von der Vorgabe „Namenlos“ in „Finden und Ersetzen“ geändert. Der Text der Title-Eigenschaft erscheint in der Titelzeile des Fensters, wenn das Fenster dargestellt wird. Folgende Abbildung zeigt das Fenster der fertigen Applikation.

Abb. 29: Der Wert der Title-Eigenschaft in der fertigen Applikation



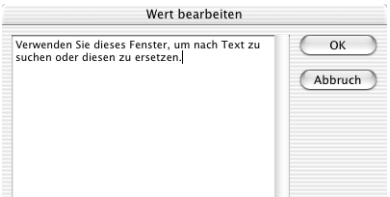
Im Eigenschaftenfenster werden Drop-Down-Menüs durch ein Dreieck markiert. Im Beispiel wird die Frame-Eigenschaft über das Drop-Down-Menü des Dokumentfensters auf beweglicher modaler Dialog geändert.

Abb. 30: Ändern der Frame-Eigenschaft



Das Icon (oder unter Windows/Mac OS Classic) zeigt an, dass nach einem Klick darauf ein Texteingabedialog erscheint. Im obigen Beispiel befindet sich bei der BalloonHelp-Eigenschaft ein solches Icon. Sie können den Hilfetext direkt in den Eingabebereich bei BalloonHelp eingeben oder durch Klick auf einen größeren Texteingabebereich aufrufen, wie er in folgender Abbildung gezeigt wird.

Abb. 31: Hilfetext im „Wert bearbeiten“-Bereich eingeben

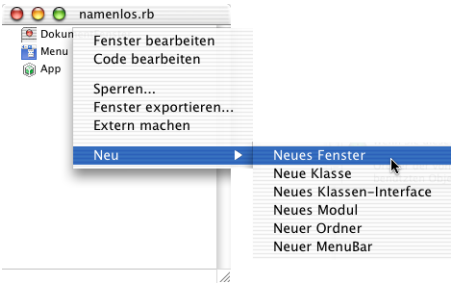


Fenster erzeugen

Wenn Sie ein neues Desktop-Applikation-Projekt beginnen, erzeugt REALbasic automatisch ein neues Fenster namens Fenster1. Es gibt zwei Möglichkeiten, Ihrem Projekt ein weiteres Fenster hinzuzufügen. Sie können einen Befehl aus der Menüleiste wählen oder das Kontextmenü des Projektfensters aufrufen.

Um mittels des Menübefehls ein neues Fenster anzulegen, klicken Sie auf **Ablage/Neues Fenster** (Windows: **Datei/Neues Fenster**). Das Kontextmenü erhalten Sie, indem Sie einen ctrl-Klick (Windows: Rechtsklick) auf das Projektfenster ausführen. Wählen Sie dann **Neu/Neues Fenster**.

Abb. 32: Neues Fenster über das Kontextmenü des Projektfensters hinzufügen



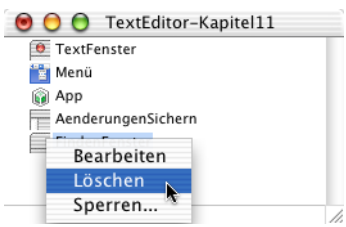
Fenster, die Sie anlegen, dienen als Vorlagen. Öffnet Ihr Programm ein Fenster, dann wird eine Kopie dieser Vorlage erzeugt. Daher kann Ihr Programm mehrere Kopien des gleichen Fensters zur gleichen Zeit öffnen. Es ist wichtig, dass man diesen Sachverhalt versteht, denn aus ihm folgt, dass man nicht zwei gleiche Fenster entwerfen muss, wenn das Programm später zwei Fenster dieser Art öffnen soll.

In der Tutorial-Applikation von REALbasic verwendet die Textverarbeitung eine Fenstervorlage als Dokumentfenster. Mit dem Menüpunkt **Datei/Neu** kann der Benutzer in der fertigen Anwendung aus der einen Vorlage beliebig viele Dokumentfenster erzeugen. Die Tutorial-Applikation benötigt zusätzliche Fenster für den „Suchen und Ersetzen"-Dialog.

Fenster entfernen

Um ein Fenster aus einem Projekt zu entfernen, klicken Sie im Projektfenster auf den Namen des Fensters und drücken die Backspace-Taste oder wählen Sie **Bearbeiten/Löschen**. Sie können auch mit gedrückter ctrl-Taste (Windows: Rechtsklick) auf den Fensternamen klicken und im Kontextmenü den Befehl **Löschen** aufrufen.

Abb. 33: Ein Fenster über das Kontextmenü entfernen



Sie können viele Aktionen in REALbasic rückgängig machen. Mit **Bearbeiten/Undo** (⌘-Z auf dem Mac, Strg und Z unter Windows) können Sie z.B. ein versehentlich gelöschtes Fenster wiederherstellen.

Vorgabefenster festlegen

Normalerweise wird das standardmäßig im Projekt vorhandene Fenster automatisch geöffnet, wenn die Applikation gestartet wird. Dieses Fenster wird auch als Vorgabefenster bezeichnet. Falls es in Ihrem Projekt mehrere Fenster gibt,

wollen Sie vielleicht, dass sich standardmäßig ein anderes oder gar kein Fenster automatisch öffnet. Dies können Sie über den Menüpunkt **Bearbeiten/Projekteinstellungen** festlegen. Gehen Sie dazu folgendermaßen vor:

1. Rufen Sie den Menüpunkt **Bearbeiten/Projekteinstellungen** auf. Es öffnet sich der Dialog „Projekteinstellungen".

Abb. 34: Der Projekteinstellungen-Dialog

2. Wählen Sie unter Vorgabefenster das gewünschte Fenster aus oder „keins", falls die Applikation kein Fenster öffnen soll.
3. Betätigen Sie den OK-Knopf, um die Auswahl zu übernehmen.

Wenn Sie Ihre Applikation starten, wird standardmäßig das ausgewählte Fenster (oder keines) geöffnet.

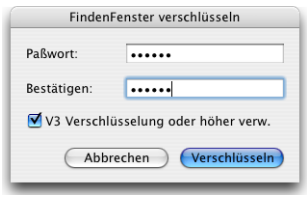
Fenster verschlüsseln

Sie können ein Fenster im Projektfenster verschlüsseln (schützen) oder entschlüsseln (freigeben). Ein gesperrtes Fenster kann nicht im Fenster-Editor angezeigt werden. Niemand kann auf den Quelltext eines verschlüsselten Fensters und seiner Steuerelemente zugreifen.

Beim Verschlüsseln müssen Sie ein Passwort festlegen, das später wieder zum Entschlüsseln verwendet werden kann. In diesem Zusammenhang können Sie auch die Checkbox **V3.0 Verschlüsselung oder höher verw.** deaktivieren, wenn Sie die Rückwärtskompatibilität zu früheren REALbasic Versionen garantieren wollen.

1. Um ein Fenster zu verschlüsseln, selektieren Sie es im Projektfenster und wählen den Menüpunkt **Bearbeiten/Verschlüsseln**. Alternativ können Sie im Projektfenster einen ctrl-Klick (Windows: Rechtsklick) auf den Fensternamen ausführen und im Kontextmenü den Menüpunkt **Verschlüsseln** aufrufen. Es erscheint folgender Dialog:

Abb. 35: Der Verschlüsseln-Dialog

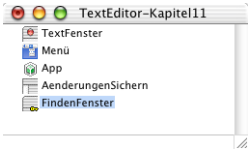


2. Geben Sie ein Passwort für die Verschlüsselung ein und bestätigen Sie dieses.
3. Selektieren Sie **V3.0 Verschlüsselung oder höher verw.** nur, wenn das Fenster mit REALbasic Version 3 oder höher verwendet wird. Wenn Sie diese Option nicht anwählen, wird der Algorithmus von Version 2.1 verwendet.

Wichtig: Vergessen Sie nicht ihr Passwort.

Ein gesperrtes Fenster erscheint im Projektfenster mit einem Schlüssel im Fenstersymbol. In folgender Abbildung wurde der Dialog FindenFenster verschlüsselt.

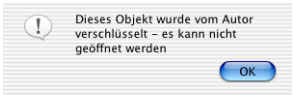
Abb. 36: Verschlüsseltes FindenFenster im Projektfenster



Hinweis: Das Symbol kennzeichnet ein verschlüsseltes, bewegliches, modales Fenster. Jeder Fenstertyp hat ein eigenes Symbol.

Wenn ein Anwender versucht, ein verschlüsseltes Fenster im Fenster-Editor zu öffnen, erscheint folgende Warnbox:

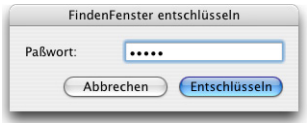
Abb. 37: Hinweis bei verschlüsselten Objekten



Wenn Sie das Fenster und den dazu gehörenden Quelltext bearbeiten wollen, müssen Sie es zunächst entschlüsseln:

1. Selektieren Sie das Fenster im Projektfenster und wählen den Menüpunkt **Bearbeiten/Entschlüsseln**. Alternativ können Sie im Projektfenster einen ctrl-Klick (Windows: Rechtsklick) auf den Fensternamen ausführen und im Kontextmenü den Menüpunkt **Entschlüsseln** aufrufen. Es erscheint der Entschlüsseln-Dialog.

Abb. 38: Der Entschlüsseln-Dialog



2. Geben Sie das Passwort ein und klicken Sie auf **Entschlüsseln**. Nach kurzer Zeit verschwindet der Schlüssel aus dem Icon des Fensters, um anzuzeigen, dass die Entschlüsselung erfolgreich war. Wenn Sie ein falsches Passwort eingegeben haben, informiert Sie eine entsprechende Dialogbox darüber. Wenn Sie das Passwort nicht kennen oder vergessen haben, gibt es keine Möglichkeit, das Fenster zu entschlüsseln.

Einfache Mitteilungsboxen

In REALbasic gibt es auch einen einfachen Weg, Dialogboxen anzuzeigen, ohne dazu ein Fenster erzeugen und mit Steuerelementen versehen zu müssen. Allerdings ist der Gestaltungsspielraum dabei stark eingeschränkt: Bei den Mitteilungsboxen handelt es sich um Fenster mit einem Symbol, etwas Text und einem oder mehreren Knöpfen. Die einzige Interaktionsmöglichkeit ist das Betätigen eines Knopfes.

Zum Erzeugen dieser Boxen gibt es die MsgBox-Funktion und die MessageDialog-Klasse.

Die MsgBox Funktion

Verwenden Sie die MsgBox-Funktion, wenn Sie eine Meldung in einem modalen Dialog anzeigen möchten. Geben Sie beim Aufruf einfach den anzuzeigenden Text an. Die folgende Meldung könnte angezeigt werden, wenn ein Datei-Upload abgeschlossen ist.

```
MsgBox "Dateitransfer abgeschlossen"
```

Mit dieser Zeile wird ein Dialog angezeigt, der einen Knopf enthält, mit dem der Benutzer die Meldung bestätigen kann.

Abb. 39: MsgBox unter Mac OS X

Die MsgBox-Funktion besitzt zwei optionale Parameter, mit denen Sie den Dialog leicht abändern können. So können Sie die Anzahl der angezeigten Buttons angeben, den Standard-Button bestimmen und auch ein Icon angeben.

Wenn Sie die optionalen Parameter verwenden, wird MsgBox immer den Index des gedrückten Buttons zurückgeben. Wenn Sie also mehr als einen Button anzeigen, müssen Sie den Rückgabewert auswerten.

Die MessageDialog Klasse

Verwenden Sie die MessageDialog Klasse, wenn Sie komplexere Dialoge gestalten wollen. Mit der MessageDialog Klasse können Sie Dialoge erzeugen, die bis zu drei Buttons enthalten und deren Beschriftung und Funktion bestimmen. Zusätzlich können Sie einen erklärenden Text anzeigen.

Da MessageDialog eine Klasse ist, können Sie dies nicht mit einer Quelltextzeile erreichen. Sie müssen eine Variable deklarieren, instanziieren, deren Eigenschaften setzen und den Rückgabewert auswerten, der Ihnen mitteilt, welcher Button gedrückt wurde.

Ein MessageDialog kann drei Buttons besitzen: ActionButton, CancelButton und AlternateActionButton. Diese besitzen folgende Eigenschaften:

Eigenschaft	Beschreibung
Caption	Die Beschriftung des Knopfes.
Visible	Auf True setzen, um den Knopf anzuzeigen.
Default	Auf True setzen, um den Knopf als Standardknopf zu hervorzuheben. Standardmässig ist diese Eigenschaft für die Klasse ActionButton auf True gesetzt.
Cancel	Auf True setzen, damit dieser Knopf auf das Drücken der Esc-Taste reagiert. Unter Mac OS reagiert dieser ebenso auf das Drücken von ⌘-„.". Diese Eigenschaft kann nur für einen Knopf auf True gesetzt sein. Einige Betriebssysteme erlauben nicht, dass der Cancel-Knopf gleichzeitig Standardknopf ist.

Per Voreinstellung wird nur der Standardknopf angezeigt. Sie können die anderen Knöpfe anzeigen, indem Sie deren Visible-Eigenschaft auf True setzen.

Jetzt müssen Sie nur noch den anzuzeigenden Text und das Symbol festlegen (Hinweis, Achtung, Stopp, Frage).

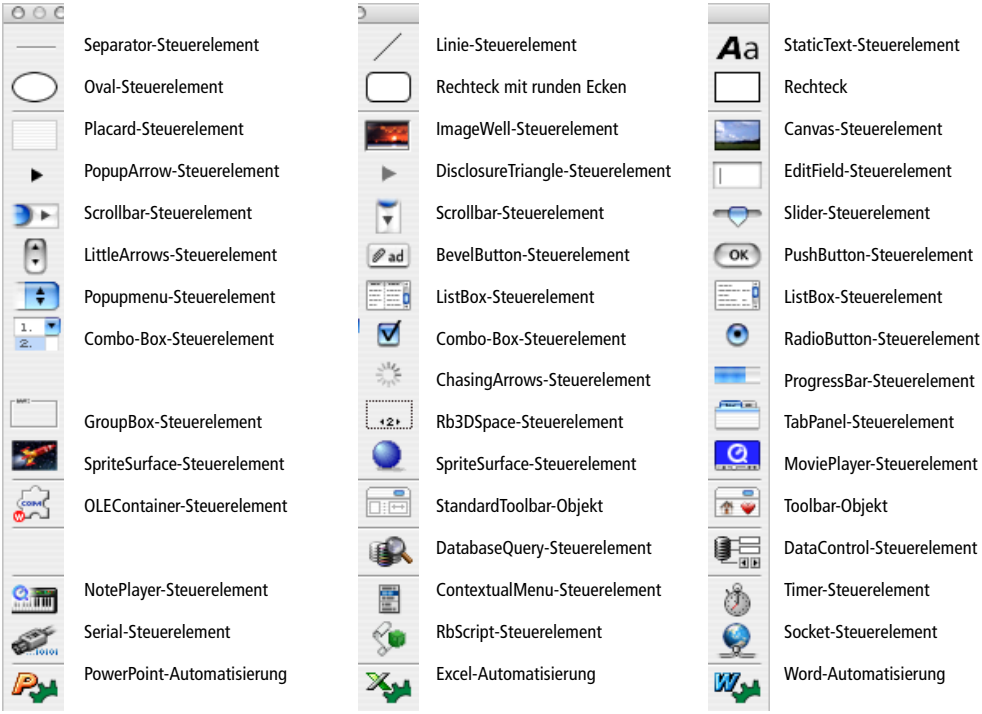
Wenn der Benutzer einen Knopf anklickt, wird ein MessageDialogButton-Objekt zurückgegeben. Dieses kann vom Typ AcceptButton, CancelButton oder AlternateActionButton sein. Anhand dieses Typs können Sie feststellen, welchen Knopf der Anwender angeklickt hat.

In der Online-Sprachreferenz finden Sie weitere Informationen und Beispiele.

Benutzerinteraktion über Steuerelemente

Informationen, die Ihr Programm vom Benutzer entgegennehmen soll, erhält es durch Steuerelemente. Dazu können die vordefinierten Steuerelemente von REALbasic verwendet werden. Es ist aber auch möglich, eigene Steuerelemente zu entwerfen. Die eingebauten Steuerelemente sind in der Steuerelementepalette zusammengefasst.

Abb. 40: Die Steuerelementepalette



Einige Steuerelemente sind später für den Anwender sichtbar (z.B. PushButtons, ListBoxen), andere nicht. Indem Sie beispielsweise ein Socket-Steuerelement auf ein Fenster ziehen, erweitern Sie Ihre Applikation um Fähigkeiten zur Netzwerkkommunikation. Dem Fenster kann man das aber nicht ansehen.

Im unteren Teil der Steuerelementepalette liegen die Steuerelemente zur Automatisierung der Microsoft Office-Programme PowerPoint, Excel und Word. Auch diese Steuerelemente enthalten keine für den Anwender sichtbaren Interface-Elemente.

Wenn Sie eigene Steuerelemente entwickeln oder Steuerelemente von Drittanbietern installieren (auch ActiveX-Steuerelemente), erscheinen diese unterhalb der Steuerelemente zur Office-Automatisierung.

Steuerelemente anpassen

Um ein Steuerelement Ihren eigenen Bedürfnissen anzupassen, erstellen Sie eine neue Klasse, die auf einem Standard-Steuerelement basiert und passen diese anschließend Ihren Wünschen an. Weitere Informationen zu diesem Thema finden Sie im Abschnitt „Beispiele von Unterklassen“ auf Seite 306.

Anordnen der Steuerelementepalette

Der rechte Knopf in der Titelleiste der Steuerelementepalette schaltet die Anzeige der Palette zwischen horizontal und vertikal um. Folgende Abbildung zeigt beide Ausrichtungen:

Abb. 41: Horizontale und vertikale Palettenausrichtung

Unter Windows läuft die REALbasic IDE in einem sogenannten MDI-Fenster (Multiple Document Interface). Normalerweise ist die Steuerelementepalette ein schwimmendes Fenster, das Sie auch aus dem MDI-Fenster herausziehen können. Sie können sie aber auch an eine der vier Seiten des MDI-Fensters andocken. Wählen Sie dazu den Menüpunkt **Fenster/Steuerelemente andocken** und die gewünschte Fensterseite aus.

Steuerelemente verwenden

Hinzufügen

Es gibt zwei Möglichkeiten, ein Steuerelement in ein Fenster zu integrieren. Entweder Sie verwenden die Steuerelementepalette oder das Kontextmenü des Fensters, dem Sie das Steuerelement hinzufügen wollen.

Wenn Sie die Steuerelementepalette verwenden:

1. Holen Sie das Fenster, auf dem das Steuerelement platziert werden soll, nach oben. Ist es nicht geöffnet, öffnen Sie es mit einem Doppelklick auf seinen Namen im Projektfenster.
2. Ziehen Sie das gewünschte Steuerelement aus der Steuerelementepalette auf das Fenster.

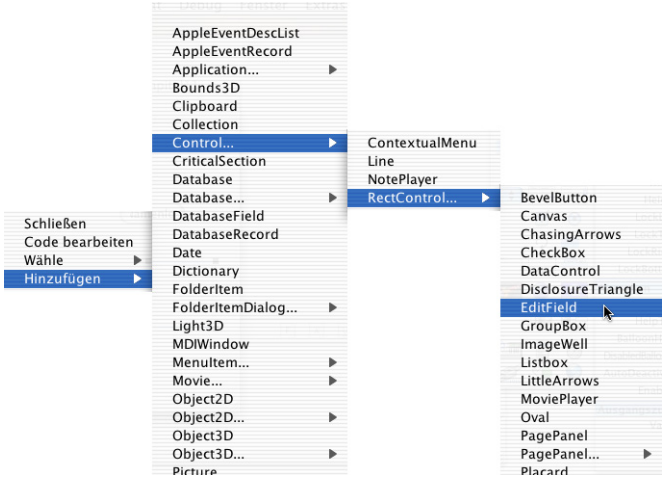
Wenn Sie das Kontextmenü verwenden:

1. Führen Sie einen ctrl-Klick (Rechtsklick) in das entsprechende Fenster aus, um das Kontextmenü aufzurufen. Der Eintrag **Steuerelement hinzufügen** öffnet ein hierarchisches Untermenü.

Abb. 42: Das Kontextmenü des Fenstereditors

2. Verwenden Sie den Menüpunkt **Steuerelement hinzufügen/Standard**, um das Steuerelement auszuwählen, das Sie hinzufügen möchten. Bei den hier aufgeführten Steuerelementen handelt es sich fast nur um sichtbare Steuerelemente. Ausnahmen sind das Serial- und das Socket-Steuerelement.

Abb. 43: Hinzufügen eines EditField-Steuerelements über das Kontextmenüs des Fenstereditors



Das **Steuerelement hinzufügen/Beliebig**-Untermenü führt alle verfügbaren Objekte auf. Wenn Sie ein Objekt auswählen, das kein Steuerelement ist, wird es in der IDE durch ein generisches Icon repräsentiert. Durch Klick auf dieses Icon wählen Sie es im Code-Editor des Fensters aus, als ob es ein sichtbares Steuerelement wäre.

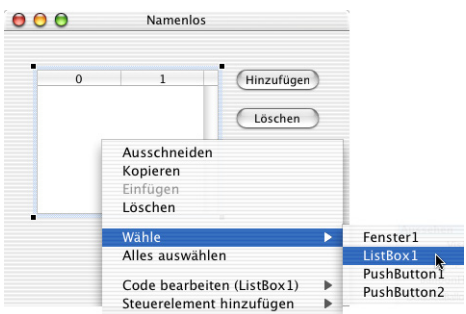
Abb. 44: Icon für ein allgemeines Steuerelement

Seine Eigenschaften können Sie im Eigenschaftenfenster modifizieren – genauso wie bei einem Steuerelement. Falls das Objekt Events besitzt, können Sie diese im Code-Editor bearbeiten.

Steuerelemente auswählen

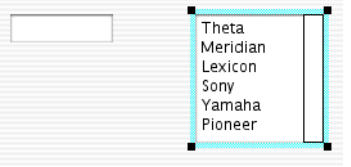
Steuerelemente können auf drei Arten selektiert werden: Mit Mausklick, der Tabulatortaste oder über das Kontextmenü. Am einfachsten wählen Sie ein Steuerelement aus, indem Sie darauf klicken. Das Kontextmenü enthält zwei Möglichkeiten, um ein Steuerelement auszuwählen, **Auswählen** und **Alles auswählen**. Über **Alles auswählen** selektieren Sie alle Steuerelemente im Fenster. Im **Auswählen**-Untermenü sind alle Steuerelemente aufgelistet, die sich im Fenster befinden. Indem Sie auf einen Eintrag klicken, wählen Sie das entsprechende Steuerelement aus.

Abb. 45: Auswahl eines Steuerelements über das Kontextmenü des Fenstereditors



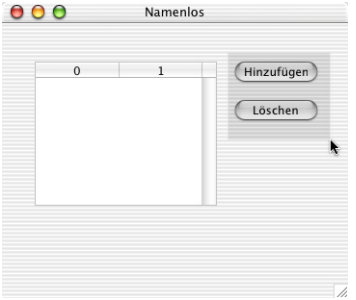
Wenn Sie ein Steuerelement selektieren, zeichnet REALbasic einen Rahmen um das Steuerelement.

Abb. 46: Eine selektierte ListBox unter Mac OS X



Sie können mit der Tabulatortaste von einem zum nächsten Steuerelement eines Fensters wechseln. Dabei kann man auch die Reihenfolge sehen, in der die Steuerelemente später vom Benutzer angesprungen werden können. Siehe auch „Ändern der Steuerelementereihenfolge (Tab-Reihenfolge)“ auf Seite 138. Indem Sie Shift-Tabulator drücken, werden die Steuerelemente in umgekehrter Reihenfolge angesteuert. Ist ein Steuerelement angewählt, erscheinen Angreifecken, mit denen die Größe des Elements geändert werden kann. Sollen mehrere Elemente ausgewählt werden, müssen sie mit gedrückter Shift-Taste mit der Maus angeklickt werden. Sie können auch einen Rahmen um eine Gruppe von Steuerelementen ziehen, um diese auszuwählen. Unter Mac OS X werden alle ausgewählten Steuerelemente von einer halbtransparenten Markierung umsäumt.

Abb. 47: Auswahl zweier Steuerelemente mit Hilfe des Auswahlrechtecks



Hinweis: Mit dem Menüpunkt **Bearbeiten/Alles auswählen** können Sie alle Steuerelemente im obersten Fenster auf einmal auswählen. Nachdem alle ausgewählt sind, können Sie mit Shift-Klick auf ein Steuerelement dieses einzelne aus der Auswahl entfernen.

Unsichtbare Steuerelemente auswählen

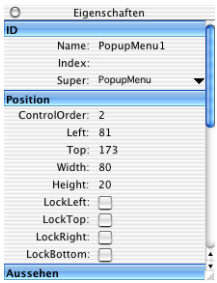
Es ist möglich, dass ein Steuerelement von der Bildfläche verschwindet, z.B. wenn Sie ihm einen negativen Wert bei der **Left**- oder **Top**-Eigenschaft zugewiesen haben (dann liegt es links neben oder über dem Fenster). Oder wenn **Width** und **Height** den Wert 0 haben, dann befindet sich das Steuerelement zwar weiterhin an der gleichen Stelle, ist aber nicht sichtbar. Um ein unsichtbares Steuerelement anzuwählen, können Sie das Kontextmenü des Fensters verwenden. Das **Auswählen**-Untermenü listet immer alle Steuerelemente auf, die zum Fenster gehören, auch wenn diese nicht sichtbar sind. In der folgenden Abbildung wird ein EditField mit dem Wert 0 für **Width** und **Height** auf diese Art und Weise angewählt. Ist das Element ausgewählt, erscheint ein Auswahlrahmen und seine Eigenschaften werden im Eigenschaftenfenster angezeigt. Sie können dann das Eigenschaftenfenster dazu verwenden, die Einstellungen für Position und Größe zu ändern.

Abb. 48: Unsichtbare Steuerelemente auswählen

Ändern der Position eines Steuerelements

Die Position eines Steuerelements kann man mit der Maus, mit den Pfeiltasten (um jeweils einen Pixel) oder über die Position-Eigenschaften im Eigenschaftenfenster ändern.

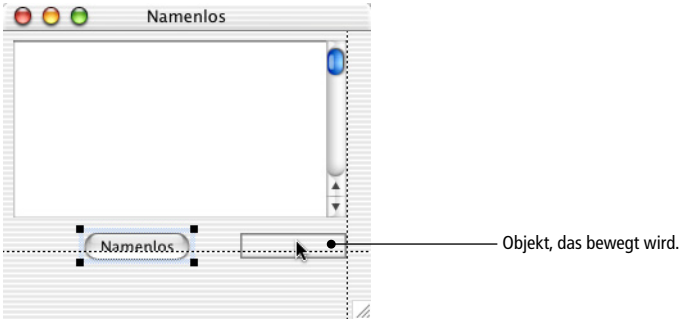
Abb. 49: Die Position-Eigenschaften eines PopupMenu-Steuerelements



Die Left- und Top-Eigenschaften legen die Position der linken oberen Ecke des Steuerelements fest, während die Width- und Height-Eigenschaften die Größe definieren. Sie können diese Eigenschaften immer verwenden, um Position und Größe eines Steuerelements präzise festzulegen.

Wenn Sie ein Steuerelement draggen, können Sie es an anderen Objekten im Fenster ausrichten, indem Sie auf die automatisch erscheinenden vertikalen und horizontalen Hilfslinien achten. Wenn sich das Objekt, das Sie draggen, in Höhe der horizontalen oder vertikalen Kante eines anderen Objekts befindet, erscheinen vorübergehend Hilfslinien, die es erlauben, das Objekt präzise zu positionieren. Folgende Abbildung zeigt, wie man einen PushButton in Bezug auf die Grundlinien eines anderen PushButtons und den rechten Rand einer ListBox ausrichtet.

Abb. 50: Ausrichten eines Steuerelements an den Hilfslinien:

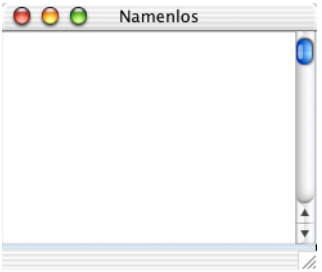


Position eines Steuerelements mit der Lock-Eigenschaft festlegen

Jedes sichtbare Steuerelement hat die vier boole'sche Eigenschaften LockLeft, LockRight, LockTop und LockBottom, die Sie verwenden können, um die horizontalen und vertikalen Kanten mit den zugehörigen Kanten des Fensters fest zu verbinden. Wenn eine dieser Eigenschaften eingeschaltet ist, bleibt der Abstand zwischen der Kante des Steuerelements und der zugehörigen Fensterkante immer gleich, wenn der Anwender die Größe des Fensters ändert.

Diese Eigenschaften werden verwendet, um REALbasic mitzuteilen, dass es die Steuerelemente skalieren oder bewegen muss, wenn der Anwender die Größe des Fensters ändert. Wenn Sie z.B. ein MultiLine-EditField für ein Textverarbeitungsfenster verwenden, wollen Sie die Kanten des Steuerelements mit den Kanten des Fensters gleichsetzen und setzen dafür die Eigenschaften LockLeft, LockRight, LockTop und LockBottom, damit sich das Steuerelement automatisch anpasst, wenn der Anwender die Größe des Fensters ändert. Folgende Abbildungen demonstrieren dies.

Abb. 51: Ein MultiLine-EditField für eine Textverarbeitung



Folgende Abbildung zeigt, wie die vier Eigenschaften funktionieren.

Abb. 52: Ändern der Größe des Fensters mit aktiven und inaktiven Lock-Eigenschaften

Lock-Eigenschaften ausgeschaltet. Das EditField behält seine Größe und Position, wenn sich die Fenstergröße ändert.

Lock-Eigenschaften eingeschaltet. Das EditField wächst und schrumpft, wenn sich die Fenstergröße ändert.

Im nächsten Beispiel sind nur die Eigenschaften LockRight und LockBottom gesetzt. Das EditField wandert nach unten und nach rechts, wenn das Fenster vergrößert wird.

Abb. 53: Ändern der Größe des Fensters, wenn LockLeft und LockTop nicht aktiv sind

Nur die untere und die rechte Seite des Steuerelements sind „gelockt".

Interessant wird es, wenn wir ein weiteres Steuerelement hinzufügen. Sehen Sie folgende Abbildung:

Abb. 54: Ein PushButton unter dem EditField

Der PushButton sollte nur die Eigenschaften LockRight und LockBottom aktiviert haben. Er soll seine relative Position zum EditField behalten. Er soll aber nicht seine Größe ändern, wenn sich die Größe des Fensters ändert. Dies zeigt folgende Abbildung.

Abb. 55: Effekt beim Ändern der Fenstergröße, wenn ein PushButton „gelockt" ist

Der PushButton hat die Eigenschaften LockRight und LockBottom. Er behält Form und relative Position in Bezug auf das EditField.

Der PushButton hat alle vier Lock-Eigenschaften gesetzt. Er behält den horizontalen und vertikalen Abstand zu den Fensterrändern und wächst somit mit dem Fenster.

Ein ungewöhnlicher Effekt tritt auf, wenn der Anwender das Fenster horizontal oder vertikal verkleinert und der PushButton mit den beiden Kanten verbunden ist, die sich aufeinander zubewegen. In der folgenden Abbildung wird dies mit einem PushButton demonstriert, bei dem LockTop und LockBottom gesetzt sind. Da der PushButton den Abstand nach oben und unten beibehalten soll, sieht es so aus, als würde er beim Verkleinern des Fensters verschwinden. In folgender Bilderreihe versucht das Steuerelement seine Distanz nach oben und unten gleichzeitig zu bewahren, während sich die Höhe des Fensters verringert.

Abb. 56: Effekt beim Reduzieren der Höhe eines Fensters auf ein Steuerelement, bei dem LockTop und LockBottom aktiv ist

klein... kleiner... weg...

Dies passiert, da es Platz über- und unterhalb des PushButtons gibt. Das EditField verschwindet dagegen nicht, da es keinen Platz zwischen dem oberen Rand des EditFields und dem oberen Rand des Fensters gibt.

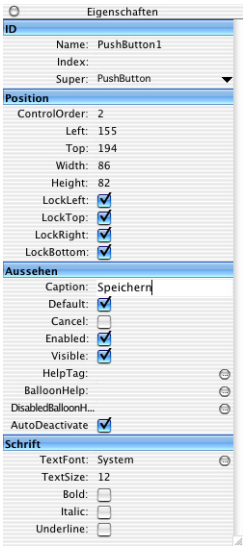
Ändern der Eigenschaften eines Steuerelements

Einige Änderungen am Verhalten von Steuerelementen müssen im Eigenschaftenfenster vorgenommen werden. Die Position des Elements lässt sich zwar ganz einfach durch Ziehen mit der Maus ändern, die meisten anderen Änderungen können aber nur im Eigenschaftenfenster eingegeben werden.

Das Eigenschaftenfenster zeigt alle Eigenschaften des gerade selektierten Steuerelements, die *innerhalb der Entwicklungsumgebung* geändert werden können. Sind mehrere Elemente ausgewählt, so werden nur die Eigenschaften gezeigt, die alle gewählten Elemente gemeinsam haben.

Einige Eigenschaften werden durch Tastatureingaben festgelegt, andere durch Aktivieren bzw. Deaktivieren von Checkboxen. Bei Eigenschaften, die durch eine Tastatureingabe gesetzt werden, können Sie den neuen Wert mit Enter oder Return bestätigen. Die Caption- oder Text-Eigenschaft einiger Steuerelemente kann bearbeitet werden, indem man das Steuerelement im Fenstereditor selektiert und den neuen Text im Eigenschaftenfenster eingibt.

Abb. 57: Ändern der Caption-Eigenschaft eines PushButtons



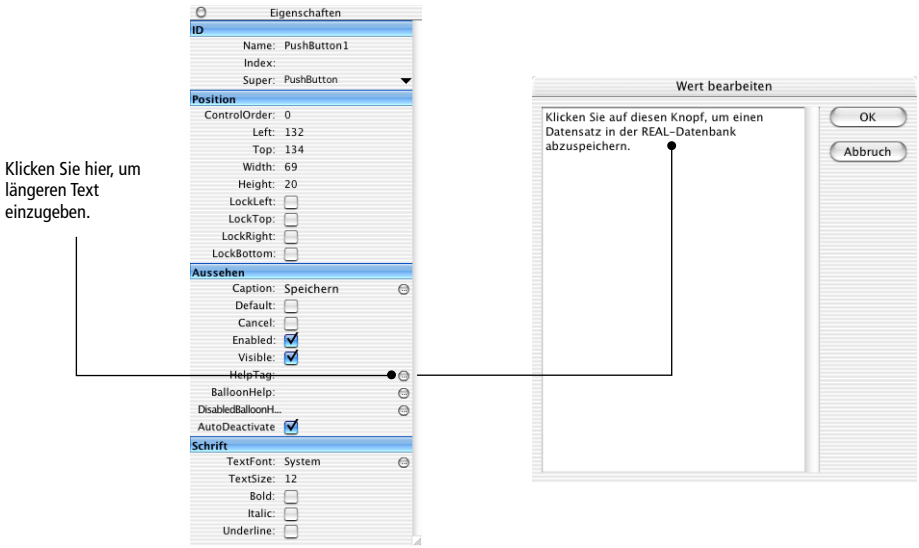
Numerische Eigenschaften (wie z.B. Left, Top, Width und Height) können als Zahlen oder als Formel eingegeben werden, deren Ergebnis eine Zahl ist. Für Formeln stehen Ihnen folgende Operatoren zur Auswahl: **+**, **-**, *****, **/**, **** (Integer-Division), **%** (Modulo) und **^**(Potenz). Sie können auch Klammern verwenden, um festzulegen, in welcher Reihenfolge

Unterfunktionen ausgeführt werden sollen. Sie können auch über Namen auf Eigenschaften referenzieren, was Ihnen erlaubt, Funktionen wie **Top*2** oder **Width+Left** zu schreiben. Wenn eine Formel ungültig ist, wird der aktuelle Wert der Eigenschaft eingesetzt. Dies erlaubt es Ihnen (z. B.) den Ausdruck ***2** als praktische Abkürzung für eine Verdoppelung des aktuellen Wertes einzugeben, wenn Sie gleichzeitig mehrere Objekte ausgewählt haben. Mit **+ +Wert** können Sie zum aktuellen Eigenschaftswert einen Wert hinzuaddieren. Um beispielsweise 10 zu addieren, verwenden Sie **+ +10**.

Texteigenschaften

Der Text eines EditField oder die Beschriftung eines PushButtons können einfach durch Eingabe im Textbereich gesetzt werden.Wenn der Text, den Sie benötigen, zu lang ist, können Sie auf das Text-Icon klicken, um ein modales Fenster aufzurufen, in das Sie längere Texte eingeben können. Ein typisches Eigenschaftenfenster sehen Sie in folgender Abbildung.

Abb. 58: Text für die Ballon-Hilfe in einem separaten Fenster eingeben



Konstanten

Sie können im Eigenschaftenfenster als Wert für eine Eigenschaft auch eine Konstante verwenden. Damit können Sie z.B. sicher stellen, dass in Ihrer Applikation alle Knöpfe eines bestimmten Typs identisch beschriftet sind. Falls Sie später diese Beschriftung ändern wollen, müssen Sie nicht jeden einzelnen Knopf ändern, sondern es reicht aus, die Konstante zu modifizieren.

Konstanten eignen sich auch zum Erzeugen mehrsprachiger Applikationen, da REALbasic einer Konstanten abhängig von der Sprache automatisch verschiedene Werte zuweisen kann.

Um eine Konstante zu verwenden, tragen Sie in das Eingabefeld der entsprechenden Eigenschaft ein Nummernzeichen „#“, direkt gefolgt vom Namen der Konstanten ein.

Im folgenden Beispiel sehen Sie die Eigenschaften eines PushButtons, für dessen Caption-Eigenschaft die Konstante „Sichern“ eingesetzt wurde.

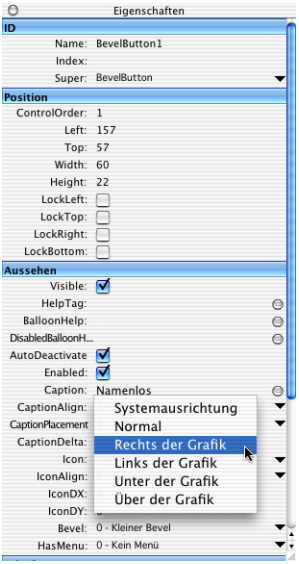
Abb. 59: Festlegen der Caption-Eigenschaft eines PushButtons über eine Konstante

Wenn Sie die Applikation erzeugen, setzt der Compiler als Namen für den Knopf den Wert der Konstanten „Sichern“ ein, den Sie für die gewählte Zielplattform und -Sprache vorgegeben haben. Zielplattform und -Sprache werden in den Compiler-Einstellungen festgelegt. Mehr dazu erfahren Sie im Abschnitt „Lokalisieren mit Konstanten“ auf Seite 234.

Auswahllisten

Einige Steuerelemente gestatten die Auswahl eines festen Wertes über ein Popup-Menü. Wählen Sie einfach den gewünschten Wert aus dem Popup-Menü aus. In folgender Abbildung wird der CaptionPlacement-Eigenschaft eines BevelButton-Steuerelements der Wert **Rechts der Grafik** zugewiesen.

Abb. 60: Einen Wert für eine Eigenschaft aus einem Popup auswählen



Farbeigenschaften

Farbeigenschaften zeigen die Farbe an. Diese kann geändert werden, indem man das Farbfeld anklickt und in der Farbauswahl eine andere Farbe wählt oder eine Farbe aus dem Farbfenster zieht und auf das Farbfeld fallen lässt. So besitzen die Steuerelemente Rechteck und abgerundetes Rechteck Eigenschaften, die den Rand und die Füllfarbe kontrollieren. Sie können diese Farben einstellen, indem Sie im Eigenschaftenfenster auf die entsprechende Farbe klicken und eine andere zuweisen.

Abb. 61: Farbe im Eigenschaftenfenster auswählen

Steuerelemente löschen

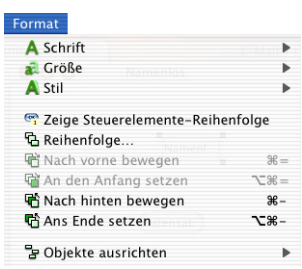
Sie können ein Steuerelement entweder über das Kontextmenü des jeweiligen Steuerelements oder über einen Menübefehl aus einem Fenster entfernen.

1. Holen Sie das Fenster, auf dem das Steuerelement platziert werden soll, nach oben. Ist es nicht geöffnet, so öffnen Sie es mit einem Doppelklick auf seinen Namen im Projektfenster.
2. Klicken Sie auf das zu löschende Steuerelement, um es zu selektieren.
3. Löschen Sie das Steuerelement mit **Bearbeiten/Ausschneiden**, **Bearbeiten/Löschen** oder Backspace. Oder führen Sie einen ctrl-Klick (Windows: Rechtsklick) auf dem Steuerelement aus und wählen **Löschen** aus den Kontextmenü. Falls Sie eine Kopie des gelöschten Steuerelements in der Zwischenablage ablegen möchten, wählen Sie **Ausschneiden** anstelle von **Löschen.**

Die Ebenen

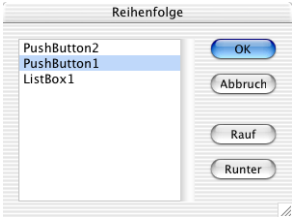
Jedes Steuerelement liegt in einer Ebene (Layer), die man sich wie eine Folie aus durchsichtigem Plastik vorstellen kann, auf der das Steuerelement liegt. Diese „Folien" liegen alle übereinander und die Reihenfolge bestimmt, welches Element über einem anderen liegt. Das Format-Menü enthält Kommandos, mit denen Elemente innerhalb dieser Ebenen nach oben und nach unten bewegt werden können, so dass sich die Zeichenreihenfolge verändert. Diese Reihenfolge spielt nur dann eine Rolle, wenn sich Steuerelemente überdecken.

Abb. 62: Das Format-Menü



Wenn Sie Steuerelemente auf eine GroupBox oder ein TabPanel legen, muss die GroupBox oder das TabPanel unter den anderen Elementen liegen, da sonst die anderen Elemente nicht sichtbar sind. Die Ebenen bestimmen auch die Reihenfolge, in der die Steuerelemente beim Betätigen der Tabulatortaste ausgewählt werden. Allerdings müssen Sie die Ebenen nicht verändern, um die Reihenfolge, in der sie angesprungen werden, zu bestimmen. Stattdessen können Sie den Menüpunkt **Bearbeiten/Reihenfolge** verwenden.

Abb. 63: Der Reihenfolge-Dialog



Der Fokus

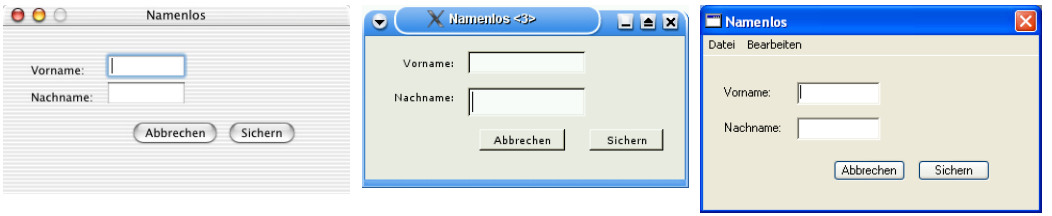
Der Fokus ist eine visuelle Markierung, die dem Anwender mitteilt, welches Steuerelement Tastatureingaben empfängt. Unter Mac OS kann der Fokus nur auf den Steuerelementen EditField, ComboBox, Canvas, PushButton und ListBox liegen. Unter Windows kann der Fokus zusätzlich auf Slider-, PopupMenu- und Checkbox-Steuerelementen liegen. Unter Linux können EditField, ComboBox, CheckBox, PushButton, PopupMenu und Slider den Fokus bekommen.

EditFields

Wenn der Fokus auf einem EditField liegt erscheint ein blinkender Cursor, der Texteingaben entgegennimmt. Das Verhalten des EditFields bei gedrückter Tabulatortaste wird durch die AcceptTabs-Eigenschaft geregelt. Wenn diese Eigenschaft den Wert False hat, bewirkt das Drücken der Tabulatortaste, dass das nächste Steuerelement in der Reihenfolge den Fokus erhält. Wenn die AcceptTabs-Eigenschaft den Wert True besitzt, akzeptiert das EditField das Tabulatorzeichen genau wie jeden Buchstaben als Texteingabe und behält den Fokus.

In der folgenden Abbildung liegt der Fokus auf dem EditField „Vorname“. Anders als unter Mac OS X erhält das EditField unter Windows und Linux keine Fokusmarkierung. Lediglich die blinkende Einfügemarke deutet auf den Fokus hin.

Abb. 64: EditFields - einmal mit und einmal ohne Fokus



ComboBoxen

Wenn der Fokus auf einer ComboBox liegt, wird der ausgewählte Eintrag hervorgehoben und ein blinkender Cursor angezeigt. Unter Mac OS wird zusätzlich rund um die ComboBox ein Rahmen gezeichnet. Eine ComboBox mit Fokus verhält sich wie ein EditField mit Popup-Menü. Mit den Pfeiltasten können Sie einen Eintrag auswählen und die Auswahl mit Enter bestätigen. Natürlich können Sie den ausgewählten Eintrag auch bearbeiten und ersetzen.

Abb. 65: ComboBoxen

ListBoxen

Wenn eine ListBox auf dem Mac den Fokus besitzt, zeichnet REALbasic einen Rahmen um die ListBox. Wenn unter Windows der Fokus auf einer ListBox liegt, zeichnet REALbasic entweder ein gestricheltes Rechteck um das zuvor selektierte Listenelement oder – falls kein Element selektiert war – um das erste Listenelement. Wenn das zuvor selektierte Objekt aus dem Sichtbereich gescrollt wurde, gibt es keine sichtbare Veränderung im Erscheinungsbild der ListBox.

Abb. 66: ListBox mit Fokus

Windows, keine Selektion | Windows, mit Selektion | Macintosh

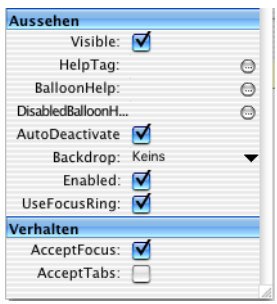
Eine Listbox reagiert auf die Pfeiltasten ↑ und ↓. Wenn Sie eine dieser Tasten drücken, ändert sich der ausgewählte Eintrag entsprechend. Die ListBox erhält auch alle anderen Tastendrücke des Anwenders. Damit können Sie eine automatische Positionierung in der Liste implementieren, die den zum gedrückten Buchstaben passenden Eintrag selektiert. Ein Beispiel für eine solche „type selection" wird mit REALbasic mitgeliefert.

Anmerkung: Wenn die ListBox das einzige Steuerelement des Fensters ist, das den Fokus erhalten kann, erhält sie diesen beim Öffnen des Fensters automatisch und behält ihn bei.

Canvas-Steuerelemente

Das Canvas-Steuerelement kann den Fokus erhalten. Deshalb bietet es sich an, eigene Steuerelemente, die ebenfalls den Fokus erhalten können sollen, auf Basis des Canvas-Steuerelements zu realisieren. So können Sie zum Beispiel das Canvas-Steuerelement dazu verwenden, andere Steuerelemente zu simulieren, die auf dem Macintosh keinen Fokus erhalten können, wie beispielweise Buttons und Popup-Menüs. Im Gegensatz zu den anderen Steuerelementen, die den Fokus erhalten können, ist die AcceptFocus-Eigenschaft des Canvas-Steuerelements standardmäßig auf FALSE gesetzt. Damit es also den Fokus erhalten kann, muss die AcceptFocus-Eigenschaft auf TRUE gesetzt werden. Dies kann man mittels Code oder über das Eigenschaftenfenster erreichen. Das Canvas-Steuerelement besitzt außerdem eine AcceptTabs-Eigenschaft, die festlegt, ob das Drücken der Tabulatortaste den Sprung zum nächsten Steuerelement im Fenster bewirkt oder das Tabulatorzeichen zur weiteren Verarbeitung an das Canvas-Steuerelement übergeben wird. Wenn die AcceptTabs-Eigenschaft FALSE ist, bewirkt das Drücken der Tabulatortaste, dass das Canvas Steuerelement den Fokus verliert und an das nächste Steuerelement weiter gibt.

Abb. 67: Fokus-relevante Eigenschaften eines Canvas Steuerelements



Falls die AcceptFocus- und UseFocusRing-Eigenschaften TRUE sind, zeigt das Canvas-Steuerelement auf dem Macintosh den Fokus als Rahmen um das Steuerelement an. Unter Mac OS Classic entspricht die Farbe den Einstellungen im Kontrollfeld „Erscheinungsbild", unter Mac OS X den Benutzereinstellungen unter „Allgemeine Einstellungen".

Abb. 68: Canvas-Steuerelemente mit und ohne Fokus

Mac OS 9 Mac OS X

Unter Windows hat UseFocusRing leider keinerlei Auswirkung. Es gibt keine automatische visuelle Rückmeldung für den Fokus. Jedoch ist es mittels Programmierung kein großes Problem, den Fokus sichtbar zu machen. In der Sprachreferenz zum Canvas-Steuerelement wird gezeigt, was dafür zu tun ist.

PopupMenu

Wenn ein PopupMenu unter Windows den Fokus bekommt, wird das derzeit selektierte Objekt hervorgehoben. Es reagiert wie die ListBox auf die Pfeiltasten und besitzt die gleiche Autolocator-Funktion. Folgende Abbildung zeigt ein Popup mit der gleichen Liste wie bei der Listbox im Beispiel zuvor, einmal mit und einmal ohne Fokus. Wenn der Anwender den Buchstaben „L“ tippt, wird der Eintrag „Leipzig“ ausgewählt; „F“ wählt Frankfurt aus usw.

Abb. 69: PopupMenu ohne und mit Fokus

Ohne Fokus Mit Fokus

CheckBox

Wenn eine CheckBox den Fokus bekommt, erscheint ein gestricheltes Rechteck um die Beschriftung der CheckBox. Wenn Sie die Leertaste drücken, während der Fokus auf der CheckBox liegt, schaltet dies den Status der CheckBox zwischen angewählt und nicht angewählt hin und her.

Abb. 70: CheckBox ohne und mit Fokus

Ohne Fokus Mit Fokus

PushButton

Wenn ein PushButton den Fokus erhält, wird ein gestricheltes Rechteck um die Beschriftung des Knopfes gezeichnet. Wenn Sie die Leertaste drücken, während der PushButton den Fokus besitzt, wird der Knopf gedrückt, das bedeutet, sein Action-Event-Handler wird ausgeführt.

Abb. 71: PushButton ohne und mit Fokus

Ohne Fokus Mit Fokus

Slider

Wenn der Focus auf einem Slider liegt, wird ein gestricheltes Rechteck um das Steuerelement gezeichnet. Das Drücken von Pfeiltaste ↑ oder ← erniedrigt den Wert des Sliders und das Drücken von Pfeiltaste ↓ oder → erhöht den Wert des Sliders. Der Betrag, um den sich der Wert bei jedem Tastendruck ändert, wird durch die LineStep-Eigenschaft des Sliders festgelegt. Normalerweise hat der Slider einen Wertebereich von 0 bis 100 und LineStep ist 1. Der Anwender kann auch

irgendwo in den Slider klicken, um den Wert direkt zu setzen. Der Betrag, um den sich der Slider bei jedem Klick bewegt, wird über die PageStep-Eigenschaft festgelegt. Die Voreinstellung für PageStep ist 20.

Abb. 72: Slider ohne und mit Fokus

Ohne Fokus Mit Fokus

ScrollBar

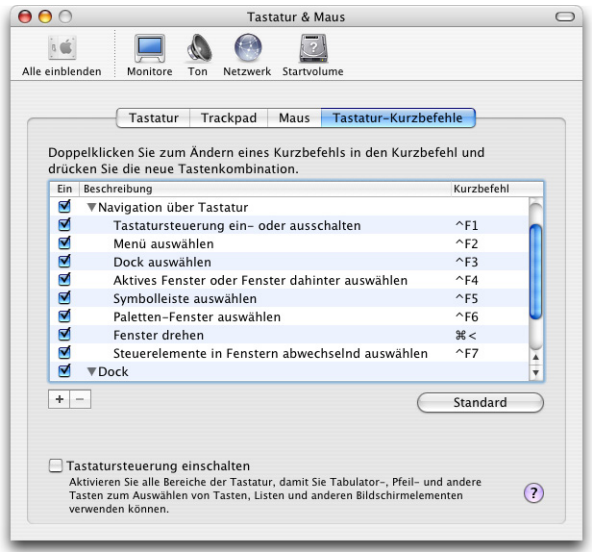
Wenn ein ScrollBar den Fokus bekommt, beginnt der Schieberegler zu pulsieren. Dies zeigt folgende Abbildung:

Abb. 73: ScrollBar ohne und mit Fokus

Ohne Fokus Mit Fokus

Tastatursteuerung

Unter Mac OS X können Sie in der „Tastatur & Maus"-Systemeinstellung die „Navigation über Tastatur" aktivieren. Diese ermöglicht dem Anwender, Programme nur mit der Tastatur zu bedienen, die normalerweise mit Tastatur und Maus bedient werden. Zum Beispiel arbeiten das Menü und das Dock standardmäßig mit Mausbewegungen. Mit aktivierter Tastatursteuerung können Sie mit den Pfeiltasten durch Menüs navigieren und den gewünschten Eintrag mit der Leertaste auswählen.



Mit aktivierter Tastatursteuerung können Sie auch Steuerelemente über die Tastatur steuern, die normalerweise keinen Fokus bekommen. So können Sie mit Hilfe der Tastatursteuerung den Wert eines RadioButtons setzen, ohne die Maus zu verwenden. Das Steuerelement, das gerade Tastatureingaben entgegen nimmt, ist von einer Aura umgeben. Mit der Tabulatortaste können Sie zum nächsten Steuerelement wechseln.

Beachten Sie, dass dies eine systemweite Einstellung ist, die vom Benutzer aktiviert werden muss. Sie können sich nicht darauf verlassen, dass diese Einstellung aktiviert ist und können diese auch nicht für den Benutzer aktivieren.

Steuerelemente duplizieren

Ein Steuerelement lässt sich duplizieren, indem man es auswählt und entweder **Bearbeiten/Duplizieren** aufruft, es mit gedrückter alt-Taste verschiebt oder ⌘-D (Windows: Strg-D) drückt.

Eventuell dem Steuerelement zugeordneter Programmcode wird dabei nicht dupliziert.

Die Objekthierarchie

Da Steuerelemente Objekte sind und REALbasic eine objektorientierte Programmiersprache ist, leiten sich alle Steuerelemente von anderen Klassen ab. Unter anderem bedeutet dies, dass jedes Steuerelement Eigenschaften von der Klasse erbt, von der es abstammt. Die meisten Steuerelemente sind Unterklassen der RectControl-Klasse. Das heißt, dass jedes dieser Steuerelemente automatisch Eigenschaften der RectControl-Klasse erbt.

In der Sprachreferenz ist die Elternklasse eines Steuerelementes als übergeordnete Klasse aufgelistet und kann mittels Mausklick direkt aufgerufen werden. Um alle Eigenschaften und Methoden eines Steuerelements einzusehen, müssen Sie sich sowohl die Eigenschaften und Methoden der übergeordneten Klasse als auch die des Steuerelements ansehen. Wenn die **Super**-Klasse eines Steuerelements wiederum eine **Super**-Klasse hat, müssen Sie in der Hierarchie weiter nach oben gehen.

Die RectControl-Klasse hat die Eigenschaften Width, Height, Top und Left. Sie bestimmen Position und Größe. Da alle Steuerelemente, die sich von der RectControl-Klasse ableiten, diese Eigenschaften übernehmen, werden diese in der Sprachreferenz nicht für jeden RectControl-Typ nochmals aufgelistet.

Die Klassenhierarchie wird im Kapitel „Die Klassenhierarchie“ der Sprachreferenz abgebildet.

Sie wird auch sichtbar, wenn sie im Kontextmenü des Fenstereditors den Menüpunkt **Hinzufügen/Beliebig** aufrufen.

Auslösen von Aktionen mit Knöpfen

Es gibt vier Arten von Knöpfen, die eine Aktion auslösen: CheckBox, PushButton, BevelButton und RadioButton. Sie leiten sich von der RectControl-Klasse ab.

PushButton

Wird ein PushButton angeklickt, dann wird er eingedrückt gezeichnet, so dass der Benutzer sofort sieht, dass er jetzt eine Aktion auslöst. Daher werden PushButtons dazu verwendet, eine Aktion einzuleiten, die direkt danach auf dem Bildschirm sichtbar wird, also beispielsweise das Schließen eines Fensters oder Drucken eines Dokuments. Unter Windows kann der PushButton den Fokus erhalten.

Abb. 74: Ein PushButton in gedrücktem und ungedrücktem Zustand:

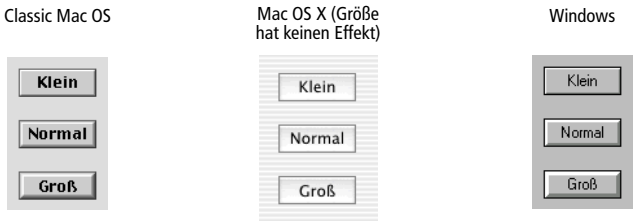
Mac OS Classic | Mac OS X | Windows

BevelButton

Das BevelButton-Steuerelement besitzt ähnliche Merkmale wie ein PushButton und einige leistungsfähige Zusatzfunktionen. Sie können eine PICT-Grafik auf dem BevelButton platzieren, die Ausrichtung und Position des Textes bzw. der Grafik beeinflussen, ein Popup-Menü hinzufügen und festlegen, welche Rückmeldung der Anwender beim Anklicken des Knopfes erhalten soll.

Abb. 75: Verschiedene BevelButtons: Als Icon, Text und „Combo“

Abb. 76: Bevel-Größen



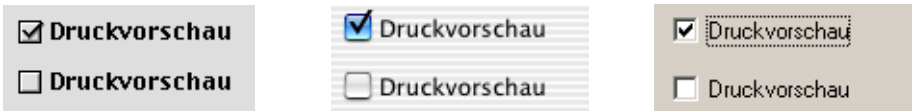
Nur unter Windows XP unterstützt die Option „Kein Bevel“ einen Mouse-Over-Effekt. Dabei wird nur die Beschriftung des Buttons dargestellt. Erst wenn der Mauszeiger in den Bereich des Buttons bewegt wird, wird der Button selbst dargestellt. Unter allen anderen Betriebssystemen entspricht die Option „Kein Bevel“ der Option „Kleiner Bevel“.

Unter Mac OS X werden vier zusätzliche Bevel-Typen unterstützt. Dies sind: abgerundet, rund, groß & rund und Aufklappdreieck. Das Aufklappdreieck wechselt beim Klicken zwischen zwei Zuständen: Spitze nach oben oder Spitze nach unten. Die anderen Bevel-Typen werden beim Klicken hervorgehoben. Unter allen anderen Betriebssystemen werden diese Bevel-Typen als „Kleiner Bevel“ dargestellt.

CheckBox

Eine CheckBox ist ein Steuerelement mit einer Zustandsvariablen, die die Werte TRUE (Häkchen wird dargestellt) und FALSE (kein Häkchen) besitzt. Sie wird zum Aktivieren oder Deaktivieren von Optionen – meist Voreinstellungen – verwendet. Das Anklicken einer CheckBox sollte nicht zu einer sofortigen Reaktion des Programms führen, mit Ausnahme des Aktivierens oder Deaktivierens anderer Steuerelemente.

Abb. 77: Eine CheckBox in den Zuständen „checked“ und „unchecked“

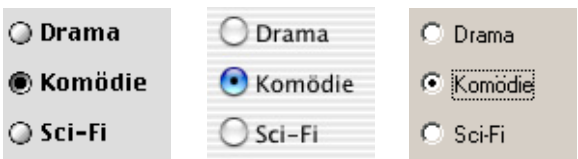


Wenn Sie genügend Platz in einer Dialogbox haben, dann sollten Sie überlegen, ob es nicht besser ist, die beiden möglichen Optionen, die die CheckBox anbietet, über ein Paar RadioButtons zu realisieren (siehe unten), da besonders für ungeübte Anwender dadurch oft klarer wird, was hier ausgewählt werden kann.

RadioButton

RadioButtons funktionieren wie die Stationswahltasten an einem alten Radio. Drückt man einen, wird er aktiviert und ein anderer dafür ausgeschaltet. Daher verwendet man RadioButtons, um dem Benutzer die Auswahl zwischen mehreren Optionen zu geben, die sich gegenseitig ausschließen. Die Knöpfe, die zusammengehören, sollten in einer Gruppe dargestellt werden.

Abb. 78: Eine Gruppe von RadioButtons, bei der ein Knopf gedrückt ist



Sollen in einem Fenster mehrere Gruppen von RadioButtons eingesetzt werden, müssen Sie diese jeweils in einem GroupBox-Steuerelement platzieren, damit die einzelnen Gruppen unabhängig voneinander reagieren (siehe „GroupBox“ auf Seite 122).

Steuerelemente zur Eingabe und zum Anzeigen von Text

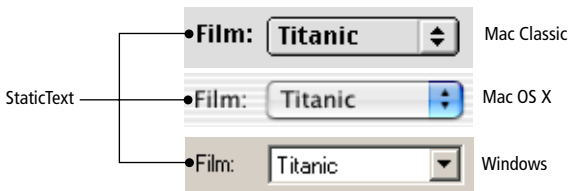
REALbasic stellt verschiedene Steuerelemente zum Anzeigen von Text zur Verfügung. Vom Typ und den Eigenschaften des gewählten Steuerelements ist es abhängig, ob der Text auch selektierbar ist oder bearbeitet werden kann.

Diese Steuerelemente leiten sich von der RectControl-Klasse ab.

StaticText

In StaticText-Objekten kann Text angezeigt werden, den der Benutzer weder selektieren noch ändern kann. Daher werden sie meist zum Beschriften von anderen Objekten verwendet. Da der Text der Beschriftung mittels Programmiercode verändert werden kann (über die Text-Eigenschaft des Objekts), kann man auch dynamischen Text erzeugen, der allerdings nur ausgelesen werden kann (read only). So können zum Beispiel schreibgeschützte Felder einer REALbasic-Datenbank auf einfache Art und Weise mit Hilfe von statischem Text erstellt werden. Sie können dies mühelos erreichen, indem Sie ein StaticText- zusammen mit einem DataControl-Steuerelement verwenden. Das DataControl-Steuerelement erlaubt es Ihnen, ein Datenbankfeld mit dem StaticText-Objekt zu verknüpfen und in diesem den aktuellen Feldinhalt anzuzeigen, während der Anwender durch die Datensätze blättert.

Abb. 79: Ein StaticText-Objekt zur Beschriftung eines Popup-Menüs:



EditField

EditFields zeigen Text an, der vom Benutzer geändert werden kann. EditFields können ein- und mehrzeilig sein. Ein mehrzeiliges EditField kann horizontale und vertikale Scrollbalken anzeigen und Text in verschiedenen Schriftarten, -stilen und -größen enthalten. Auch Absätze können – mit der StyledText-Klasse – separat linksbündig, zentriert oder rechtsbündig ausgerichtet werden. Weitere Informationen zur StyledText-Klasse finden Sie in der Sprachreferenz.

Ein einzeiliges EditField kann auch als Passwort-Feld konfiguriert werden – es zeigt dann für jeden eingegebenen Buchstaben einen schwarzen Kreis. Sie können auch Eingabemasken definieren, um die Eingabe Zeichen für Zeichen zu filtern. Zum Beispiel können Sie für ein Telefonnummern-Feld definieren, dass dieses nur Ziffern akzeptiert. Außerdem können Sie die Anzahl der erlaubten Zeichen festlegen.

Abb. 80: EditFields in einer Datenbankmaske

Abb. 81: Ein EditField mit Text in verschiedenen Schriften

ComboBox

Die ComboBox ist eine Kombination aus einen einzeiligen EditField und einem PopupMenu. Der Benutzer kann Text eingeben, aus einer Liste von Einträgen wählen oder einen Eintrag bearbeiten. Ein ComboBox-Steuerelement kann nicht als Passwortfeld verwendet werden. Auch das Definieren eines Eingabefilters ist nicht möglich.

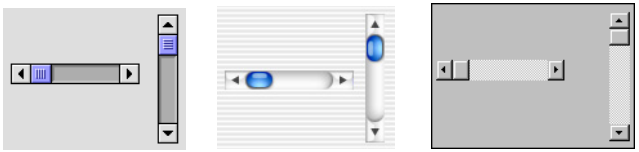
Steuerelemente zum Anzeigen und Eingaben von numerischen Werten

REALbasic verfügt über Steuerelemente, die dem Benutzer erlauben, einen Wert aus einem bestimmten Wertebereich auszuwählen. Manchmal steuern diese Steuerelemente die Anzeige eines anderen Objekts. Beispielsweise verändert ein ScrollBar-Steuerelement die Anzeige in einem EditField- oder Canvas-Objekt.

Scrollbar

Es gibt horizontale und vertikale Scrollbars. Beide Ausrichtungen finden sich in der Steuerelementepalette. Sie können ein horizontales Scrollbar-Steuerelement in die andere Orientierung drehen, indem Sie seine Größe so verändern, dass die vertikale Ausdehnung größer wird als die horizontale. Scrollbar-Steuerelemente sollten immer 16 Pixel breit sein.

Abb. 82: Horizontale und vertikale Scrollbar-Steuerelemente

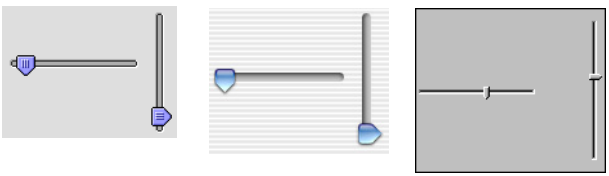


Slider

Mit Erscheinen von Mac OS 8 wurde die Macintosh-Benutzerschnittstelle um Slider erweitert. Slider funktionieren wie Scrollbar-Steuerelemente, diese werden jedoch gewöhnlich mit Textinhalten oder anderen Fensterinhalten in Verbindung gebracht und weniger mit numerischen Werten. Mit einem Slider steht nun eine intuitive Möglichkeit zum Vergrößern oder Verkleinern eines numerischen Wertes zur Verfügung.

Wie bei einem Scrollbar-Steuerelement kann man durch entsprechende Größenänderung einen horizontalen oder vertikalen Slider erzeugen. Ein Slider erhält automatisch die korrekten Proportionen, egal wie groß Sie das Slider-Objekt anlegen.

Abb. 83: Horizontale und vertikale Slider



Slider können unter Linux und Windows den Fokus erhalten. Auf diesen Plattformen werden Events ausgelöst, wenn ein Slider den Fokus erhält oder verliert.

Progressbar

Progressbars zeigen den Fortschritt einer bestimmten Aktion oder eine Kapazität (wie die Speicher- oder Festplattenbelegung) an. Anders als Scrollbars oder Slider dienen Progressbars nur zur Anzeige eines Wertes, nicht zur Eingabe. Sie

sind nur horizontal darstellbar. Ist vorher nicht genau zu bestimmen, wie lange ein Prozess dauern wird, macht man von einem geringelten Streifen Gebrauch (im Englischen „Barber Pole“ genannt, in Anlehnung an die sich drehenden geringelten Röhren, die an den Ladenschildern von Friseurgeschäften mitunter angebracht sind, um die Aufmerksamkeit von Passanten auf das Schild zu lenken).

Unter Windows wird ein Prozess, dessen Dauer man nicht voraussagen kann, in Form eines einfachen Blockes dargestellt, der sich vor und zurück bewegt.

Abb. 84: Zeitlich bestimmte bzw. unbestimmte ProgressBars

Steuerelemente für eine Auswahlliste

RadioButtons und Checkboxes können dem Benutzer eine beschränkte Auswahl an Möglichkeiten präsentieren. Manchmal ist die Verwendung von Checkboxes bzw. RadioButtons nicht möglich. Dies ist vor allem dann der Fall, wenn…

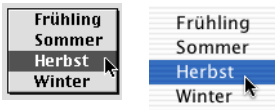
- … es sehr viele Wahlmöglichkeiten gibt.
- … sich die Wahlmöglichkeiten je nach Situation ständig ändern können.
- … mehr als eine Spalte angezeigt werden muss.

Liegt keiner dieser Fälle vor, sollten Sie möglichst mit RadioButtons oder Checkboxes arbeiten. Sie sind vor allem für ungeübte Benutzer erheblich einfacher zu handhaben, da alle Wahlmöglichkeiten zu sehen sind.

Kontextmenüs

Kontextmenüs zeigen eine Liste von Wahlmöglichkeiten in einem Menü an, wenn der Benutzer bei gedrückter ctrl-Taste (Windows: rechte Maustaste) auf ein Steuerelement klickt, das ein MouseDown-Event empfangen kann. Ein Contextual-Menu-Objekt kann dazu verwendet werden, um ein Kontextmenü für einige der sichtbaren Steuerelemente anzuzeigen.

Abb. 85: Ein Kontextmenü



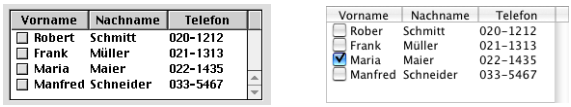
ListBox

In ListBoxen kann eine scrollbare Liste von Werten angezeigt werden. Der Benutzer kann mit der Maus oder den Pfeiltasten einen Wert auswählen. ListBoxen können mehrere Spalten enthalten, hierarchisch angelegt sein und gestatten auch die Auswahl von mehreren Einträgen gleichzeitig.

Die Steuerelementepalette hat separate Icons für ein- und mehrspaltige ListBoxes. Sie können die Anzahl der Spalten einer ListBox jedoch einfach durch Setzen der Eigenschaft ColumnCount im Eigenschaftenfenster ändern. Sie können beide Icons verwenden, um eine ListBox in einem Fenster einzufügen, und nachträglich die Anzahl der Spalten ändern.

Abb. 86: Beispiele von ListBoxen

Einzelne Spalte und mehrere Spalten

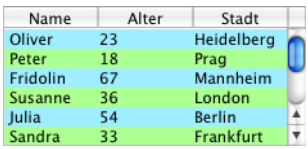


Mehrere Spalten mit Kopfzeilen und Checkboxen

Hierarchische Listbox

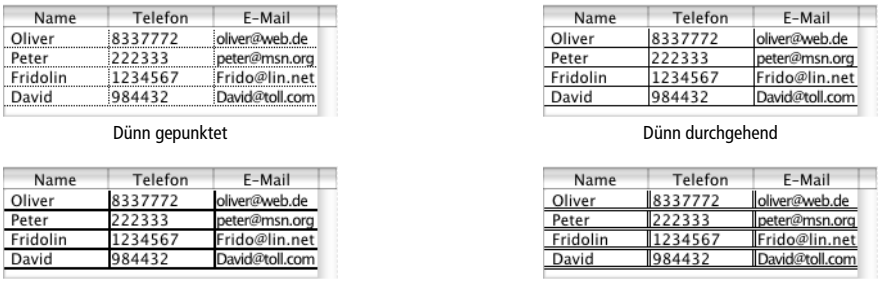
Zellen, Reihen und Spalten können mit einer Schattierung unterlegt werden. In Abbildung 87 beispielsweise sind die aufeinanderfolgenden Reihen verschiedenfarbig unterlegt. Auch die ausgewählte Reihe kann charakteristisch eingefärbt werden.

Abb. 87: Ein Beispiel für eine angepasste Schattierung



Sie können ebenfalls Anpassungen für die Begrenzungen von Zeilen und Spalten vornehmen. Abbildung 88 zeigt die vier verschiedenen horizontalen und vertikalen Begrenzungstypen, die Sie mittels einer Programmroutine zeichnen können. Die Grundeinstellung ist „None“.

Abb. 88: Die vier Stile der ListBox-Begrenzungen



Dünn gepunktet

Dünn durchgehend

Dick durchgehend

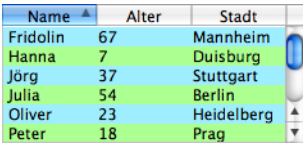
Doppelt dünn durchgehend

Sie können diese Begrenzungstypen auch auf ausgewählte Zellen oder ausgewählte Begrenzungslinien anwenden. In Abbildung 89 ist eine Zelle mit dem Begrenzungstyp „Thick Solid“ hervorgehoben, wohingegen der übrige Teil der ListBox den Begrenzungstyp „Thin Dotted“ verwendet.

Abb. 89: Eine angepasste Begrenzung um eine ausgewählte ListBox- Zelle

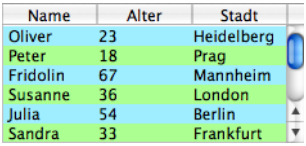
Wenn Sie einer ListBox einen Header mit Spaltenüberschrift hinzufügen, kann der Benutzer die Informationen der ListBox sortieren, indem er eine Spaltenüberschrift anklickt. Sie können die Sortierung der Reihen der ListBox auch mittels einer Programmroutine vornehmen. Die Sortierrichtung wird über einen kleinen Pfeil im Kopf-Bereich angezeigt.

Abb. 90: Eine nach der Spalte Name sortierte ListBox

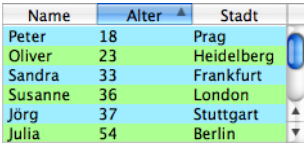


Die Standardsortiermethode arbeitet alphabetisch und liefert daher keine gültigen Ergebnisse für Zahlen und Datumseinträge. Diese Datentypen sortieren Sie mit Hilfe des **CompareRows**-Event der ListBox. Im folgenden Beispiel wird Programmiercode eingesetzt, der die Werte benachbarter Reihen in der „Alter"-Spalte vergleicht, um die gewünschte Sortierreihenfolge zu erhalten.

Abb. 91: Standardsortierung und angepasste Sortierung einer Zahlenspalte



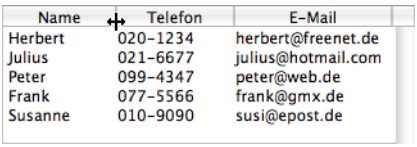
Standard-Sortierung



Angepasste Sortierung

Für mehrspaltige Listboxen können Sie festlegen, ob dem Benutzer erlaubt sein soll, die Spaltenbreite durch Verschieben der Trennlinien, zu variieren. Sie können dies Spalte für Spalte festlegen, oder aber für die gesamte ListBox. Wenn der Benutzer die Spaltenbreite verändern darf, können Sie auch die Minimal- und Maximalgröße der Spalten festlegen. Wenn die Veränderbarkeit der Spaltenbreite aktiviert ist, verwandelt sich der Mauszeiger beim Überfahren der Spaltengrenzen in ein Symbol zum Ändern der Spaltenbreite. Im folgenden Beispiel wird die Größe der Namensspalte verändert, um der Spalte mit den E-Mail-Adressen mehr Platz einzuräumen.

Abb. 92: Änderung der Breite der Spalte Name



Sie können dem Benutzer sowohl für einspaltige als auch für mehrspaltige ListBoxen erlauben, Zeilen zu draggen, um sie neu anzuordnen. Beim Draggen wird das Ziel dieser Aktion mittels einer durchgängigen Linie zwischen den Zeilen gekennzeichnet. In folgendem Beispiel wird die Zeile mit „Herbert" zwischen „Julius" und „Peter" gedraggt.

Abb. 93: Veränderung der Reihenfolge durch Draggen einer Zeile

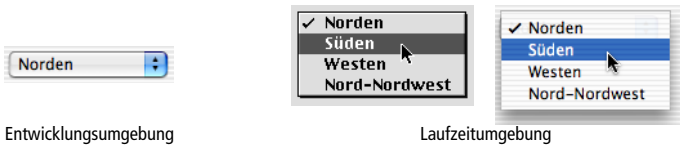
Name	Telefon	E-Mail
Herbert	020-1234	herbert@freenet.de
Julius	021-6677	julius@hotmail.com
Peter	099-4347	peter@web.de
Frank	077-5566	frank@gmx.de
Susanne	010-9090	susi@epost.de

Hierarchische ListBoxen unterstützen das Draggen von Zeilen nicht.

PopupMenu

PopupMenu-Objekte sind immer dann sehr praktisch, wenn Sie eine Spalte verschiedener Wahlmöglichkeiten benötigen, aber nur sehr wenig Platz zur Verfügung steht. Ein PopupMenu zeigt eine Auswahl von Einträgen an, aus denen der Benutzer wählen kann. Der ausgewählte Eintrag wird durch ein Häkchen hervorgehoben.

Abb. 94: Ein PopupMenu-Objekt



Entwicklungsumgebung Laufzeitumgebung

Sie können das Menü bestücken, indem Sie die Einträge im Eigenschaftenfenster eingeben oder im Programm die AddRow-Methode aufrufen, bevor das Menü angezeigt wird, z.B. im Open-Handler. Sie können die Auswahl über die Index Eigenschaft ermitteln oder auch verändern.

ComboBox

Eine ComboBox kombiniert die Eigenschaften eines EditFields und eines PopupMenus. Der Benutzer kann einen vorhandenen Eintrag wählen oder einen neuen Eintrag eingeben.

Sie können die ComboBox genauso wie das PopupMenu befüllen. Die ListIndex-Eigenschaft gibt Auskunft über den ausgewählten Eintrag. Allerdings nicht darüber, ob der Einträg editiert wurde. Sie müssen die Text-Eigenschaft auslesen, um den tatsächlichen Eintrag zu ermitteln. Der Benutzer kann über die Pfeiltasten das Menü aufklappen und Einträge auswählen. Mit der Esc-Taste wird die Auswahl abgebrochen.

BevelButton

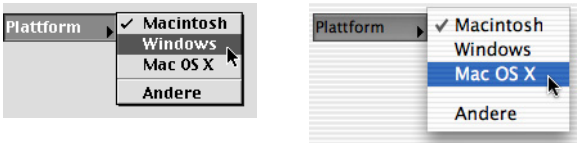
Ein BevelButton-Steuerelement kann, falls entsprechend konfiguriert, als Popup-Menü fungieren. Dazu muss die HasMenu-Eigenschaft auf 1 oder 2 gesetzt werden (normales Menü oder Menü rechts). Alle Eigenschaften des BevelButton-Steuerelements finden Sie in der REALbasic Sprachreferenz.

Folgender Code im Open-Event des BevelButtons erzeugt das BevelButton-Menü aus Abbildung 95:

```
me.captionalign=0 //Linksbündig
me.hasMenu=2      //Menü rechts
me.caption="Plattform"
me.addRow("Macintosh")
me.addRow("Windows")
me.addRow("Mac OS X")
me.addseparator
me.addRow("Andere")
```

Mit der MenuValue-Eigenschaft können Sie feststellen, welchen Menüpunkt der Anwender ausgewählt hat.

Abb. 95: Ein BevelButton mit Popup-Menü



Objekte, mit denen sich Steuerelemente zusammenfassen lassen

Wenn in einem Fenster viele verschiedene Elemente untergebracht sind, die aber nur teilweise etwas miteinander zu tun haben, dann kann das für den Benutzer sehr verwirrend sein. Daher ist es sinnvoll (und manchmal auch notwendig), Steuerelemente, die zusammengehören, zu Gruppen zusammenzufassen.

Separator

Das Separator-Steuerelement ist eine vertikale oder horizontale Linie, die Sie in einem Fenster verwenden können, um Objekte visuell zu organisieren.

Abb. 96: Ein Separator-Steuerelement

GroupBox

Eine GroupBox kann mit oder ohne Beschriftung angezeigt werden. Befinden sich mehrere Gruppen von RadioButtons in einem Fenster, müssen diese jeweils in einer GroupBox zusammengefasst werden, damit die Knöpfe der einzelnen Gruppen unabhängig voneinander funktionieren. Unter Mac OS X 10.3 scheint die GroupBox in die Fensteroberfläche eingelassen zu sein.

Abb. 97: Eine GroupBox mit und ohne Beschriftung

TabPanel

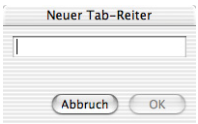
TabPanels werden verwendet, um mehrere Gruppen von Steuerelementen platzsparend unterzubringen. Jede Gruppe wird auf einer separaten Karteikarte gezeigt. Klickt der Benutzer auf einen Karteireiter, werden die derzeit angezeigten Steuerelemente verborgen und diejenigen angezeigt, die sich auf der angeklickten Karteikarte befinden.

Abb. 98: Ein TabPanel mit zwei Karteireitern:

Mit dem TabPanel-Editor fügen Sie Karteireiter hinzu und ändern deren Reihenfolge und Beschriftung. Klicken Sie auf den letzten Tab, der immer „…“ heißt, um den Editor aufzurufen.

Hier können Sie folgende Aktionen vornehmen:

- Den ersten Karteireiter umbenennen, indem Sie ihn selektieren und auf „Ändern“ klicken.
- Mit „Neu“ einen neuen Karteireiter hinzufügen und benennen.



- Eine Karteikarte auswählen und mit Klick auf „Löschen“ vom Tab-Panel entfernen.
- Die Reihenfolge der Karteireiter ändern, indem Sie einen anklicken und mit den Knöpfen „Rauf“ und „Runter“ an eine andere Position verschieben.

Nachdem Sie die gewünschten Karteireiter erzeugt haben, können Sie jeder Karteikarte Steuerelemente hinzufügen. Selektieren Sie einen Karteireiter und draggen dann die benötigten Steuerelemente auf diese Karteikarte. Wiederholen Sie dies für jede Karteikarte.

Mit der Facing-Eigenschaft können Sie die Karteireiter an jeder Kante des Steuerelements anbringen. Folgende Abbildung zeigt die Ausrichtung an der unteren Kante:

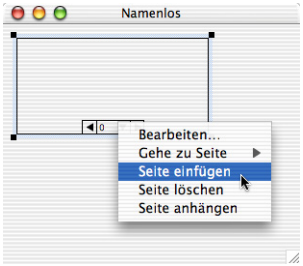
PagePanel

Das PagePanel entspricht von der Idee her dem TabPanel, mit dem Unterschied, dass der Anwender es nicht zu Gesicht bekommt. Es besitzt keine Karteireiter und hat keine sichtbare Umrandung. Lediglich die Steuerelemente auf jeder Seite sind sichtbar. Sie sind dafür verantwortlich, die entsprechende Methode zum Navigieren durch die Seiten zur Verfügung zu stellen. Sie können dies dadurch erreichen, dass Sie den Wert der Value-Eigenschaft per Programmcode festlegen. In der IDE steuern sie das Anhängen, Einfügen, Löschen und Navigieren zwischen den Seiten eines PagePanels über die Schaltflächen am unteren Rand des Steuerelements.

Verwenden Sie den Befehl **Seite einfügen** oder **Seite anhängen**, um Seiten hinzuzufügen. Oder verwenden Sie den Befehl **Seite löschen**, um die aktuelle Seite zu löschen.

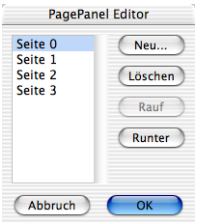
Es gibt zwei Möglichkeiten, zwischen den Seiten zu navigieren. Der **Gehe zu Seite**-Befehl zeigt ein Untermenü mit allen zur Verfügung stehenden Seitennummern an. Wählen Sie aus diesem Menü eine Seitenzahl aus, um die entsprechende Seite direkt anzuspringen. Oder klicken Sie auf die Pfeile links und links neben der Seitenzahl, um eine Seite zurück oder vor zu blättern.

Abb. 99: Das PagePanel-Popup-Menü (IDE)



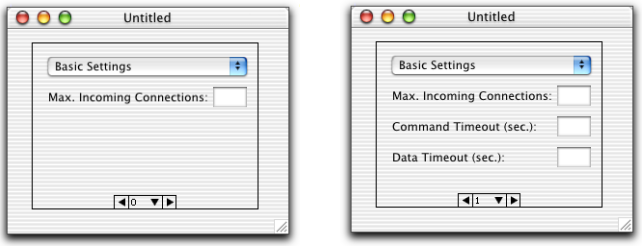
Der **Bearbeiten**-Befehl in der obigen Abbildung ruft den PagePanel-Editor auf. Der PagePanel-Editor besitzt die gleiche Benutzeroberfläche wie der TabPanel-Editor, mit dem einzigen Unterschied, dass Sie PagePanels keinen Namen zuordnen können. Mit dieser einzigen Ausnahmen funktioniert der PagePanel-Editor genauso wie der TabPanel-Editor.

Abb. 100: Der PagePanel-Editor



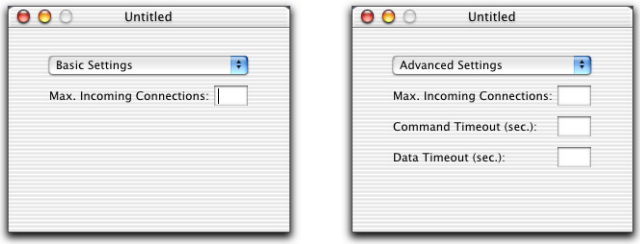
Hier ist ein einfaches Beispiel für den Einsatz eines PagePanel-Steuerelements. Der Benutzer möchte zwei verschiedene Versionen einer Benutzeroberfläche anzeigen: Eine „Basis"-Version, die standardmäßig angezeigt wird und eine „Erweiterte" Variante, die nur dann angezeigt wird, wenn der Benutzer das ausdrücklich wünscht. Zur Kontrolle, welche Seite gerade angezeigt wird, dient ein Popup-Menü.

Abb. 101: Die Seiten mit den „Basis"- und den „Erweiterten"- Einstellungen in der IDE



In der fertigen Anwendung sehen die zwei Seiten folgendermaßen aus:

Abb. 102: Die Seiten mit den „Basis“- und den „Erweiterten“- Einstellungen im laufenden Programm



Der Code im Change-Event jedes Popup-Menüs legt die Seite fest, die angezeigt werden soll, sowie die Einstellungen für das Popup-Menü der anderen Seite. Der Code des Popup-Menüs für die „Basis“-Version der Seite sieht z.B. folgendermaßen aus:

```
pagePanel1.value=me.listindex
popupMenu2.listindex=me.listindex
```

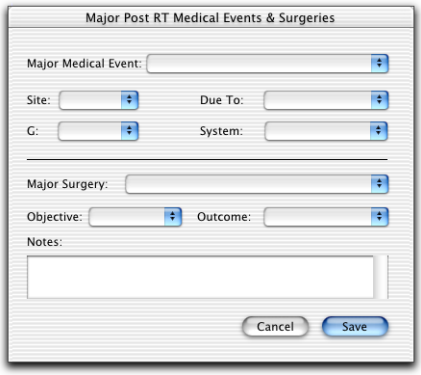
Steuerelemente zum Anzeigen von Grafik und Bildern

Sie können mit REALbasic Bilder aus anderen Programmen laden und darstellen oder mit den eingebauten Funktionen neue Grafiken zeichnen.

Line

Zeichnet eine Linie in beliebiger Länge, Breite, Farbe und in beliebiger Richtung. Die Vorgabe ist 100 Pixel Länge, ein Pixel Breite, schwarz und waagerecht.

Abb. 103: Ein Line-Steuerelement, das zum Trennen von zwei Bereichen in einer Dialogbox verwendet wurde



Rectangle

Zeichnet ein Rechteck beliebiger Länge, Breite, Randfarbe und Füllfarbe. Vorgegebene Rechtecke sind 100 Pixel breit und hoch, haben einen schwarzen Rand und sind weiß gefüllt. Da man die Farben der Ränder unabhängig voneinander bestimmen kann, ist es sehr leicht, Rechtecke zu erzeugen, die versenkt oder angehoben erscheinen. Ein Beispiel finden Sie in der Sprachreferenz.

Abb. 104: Ein Rectangle in Vorgabedarstellung, versenkt und angehoben

RoundRectangle

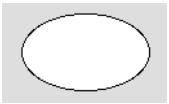
RoundRectangles ähneln Rectangles, haben aber abgerundete Ecken. Die Farbe des Randes kann hier aber nicht für die einzelnen Seiten bestimmt werden, da der Rand aus einer durchgezogenen Linie besteht. Dafür kann man Breite und Höhe der Bögen für die abgerundeten Ecken bestimmen.

Abb. 105: Ein RoundRectangle

Oval

Zeichnet ein Oval mit einer ein Pixel starken Linie, einem schwarzen Rand und einer weißen Innenfläche. Alle diese Eigenschaften können verändert werden. Die Größe des Ovals wird über die Breite und die Höhe bestimmt. Sind diese gleich groß, entsteht ein Kreis.

Abb. 106: Ein Oval



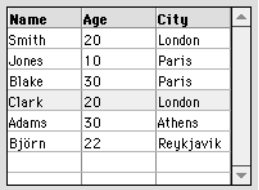
Canvas

Ein Canvas (Leinwand)-Steuerelement kann dazu verwendet werden, ein Bild, das aus einer Datei geladen wurde, darzustellen, oder um mit REALbasic-eigenen Routinen eine Grafik zu zeichnen. Wenn Sie für Ihre Benutzeroberfläche ein Bedienelement verwenden möchten, das nicht standardmäßig in REALbasic eingebaut ist, können Sie dieses ebenfalls über ein Canvas-Control und entsprechende Zeichenkommandos anlegen. Das Canvas-Steuerelement kann den Fokus erhalten, so dass Sie damit jedes beliebige Steuerelement, das den Fokus erhalten soll, simulieren können.

Abb. 107: Ein selbst erzeugtes Canvas-Steuerelement

Canvas-Steuerelemente können dazu benutzt werden, ziemlich ausgefeilte eigene Steuerelemente zu erzeugen. Die Abbildung zeigt als Beispiel eine Tabelle mit Zeilen, die selektiert werden können und Spalten, deren Sortierung geändert werden kann, indem man den Spaltentitel anklickt.

Abb. 108: Ein Canvas-Steuerelement für Fortgeschrittene



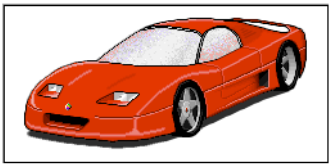
Name	Age	City
Smith	20	London
Jones	10	Paris
Blake	30	Paris
Clark	20	London
Adams	30	Athens
Björn	22	Reykjavik

Die beiden abgebildeten Steuerelemente wurden übrigens von Björn Eiríksson entworfen.

ImageWell

Das ImageWell-Steuerelement stellt einen Bereich zur Verfügung, in dem ein PICT-Bild angezeigt werden kann. Es ist relativ einfach, ein ImageWell-Steuerelement so zu programmieren, dass es per Drag & Drop übergebene Grafiken akzeptiert.

Abb. 109: Ein ImageWell-Steuerelement



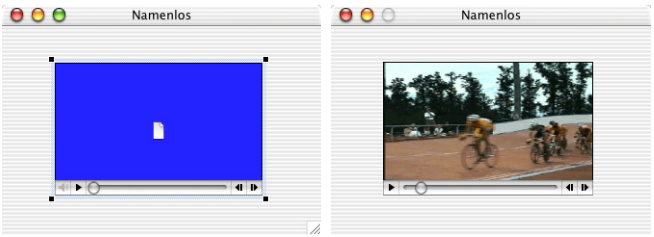
Steuerelemente zum Abspielen von Filmen, Musik, Animationen

Sofern QuickTime™ auf dem Computer des Benutzers installiert ist, kann Ihr Programm QuickTime-Filme zeigen oder die QuickTime-Instrumente verwenden, um Musikstücke zu spielen.

MoviePlayer

Das MoviePlayer-Steuerelement zeigt einen normalen QuickTime-Movie-Player.

Abb. 110: Das MoviePlayer-Steuerelement in der IDE und eine laufende Applikation



In der IDE können Sie einen QuickTime-Film wählen, der diesem Element zugeordnet ist. Sie können das Aussehen des MoviePlayers selbst bestimmen. Dabei legen Sie fest, ob der MoviePlayer komplett gezeigt wird oder zunächst nur in der Badge-Darstellung erscheint (als kleines Icon, das nach Anklicken den eigentlichen Player zeigt) und ob alle Bedienelemente des Players dargestellt werden sollen oder ob diese nicht angezeigt werden.

Es ist sehr einfach, einem MoviePlayer-Steuerelement einen QuickTime-Film zuzuordnen. Als erstes müssen Sie den QuickTime-Film ins Projektfenster ziehen. Im Eigenschaftenfenster des MoviePlayer-Steuerelements, das Sie zuvor im Fenstereditor selektieren müssen, weisen Sie nun der Movie-Eigenschaft über das Movie-Popup-Menü den Film zu, wie in Abbildung 111 gezeigt.

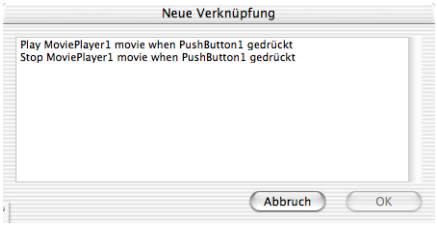
Abb. 111: So weisen Sie der Movie-Eigenschaft einen Film zu

Projektfenster Eigenschaftenfenster

Als nächstes können Sie auf dem Fenster Stopp- und Play-Buttons anlegen und diesen mittels Objekt-Verknüpfung entsprechende Aktionen zuweisen. Gehen Sie dazu folgendermaßen vor:

Legen Sie einen PushButton auf dem Fenster ab. Drücken Sie nun gleichzeitig die Shift- und Befehlstaste (Windows: Shift- und Strg-Taste) und ziehen Sie mit der Maus eine Linie vom PushButton zum MoviePlayer-Steuerelement. Es erscheint die Dialogbox „Neue Verknüpfung“, in der Sie die Möglichkeit haben, eine Aktion auszuwählen:

Abb. 112: Verknüpfen des PushButtons mit dem MoviePlayer-Steuerelement



Wählen Sie die Aktion **Play MoviePlayer1 movie when PushButton1 gedrückt**. Als nächstes legen Sie einen weiteren PushButton auf dem Fenster an und weisen diesem die **Stop MoviePlayer1 Movie**-Verknüpfung zu. Als Ergebnis erhalten Sie den in Abbildung 110 gezeigten voll funktionsfähigen MoviePlayer.

Der MoviePlayer und andere von Quicktime abhängige Features sind unter Linux nicht verfügbar.

NotePlayer

Der NotePlayer wird zwar mit einem Icon angezeigt, wenn man ihn in der Entwicklungsumgebung auf ein Fenster zieht, hat aber keine sichtbaren Interface-Elemente. Er wird dazu benutzt, um QuickTime-Instrumente zur Wiedergabe von Musik anzusteuern. Siehe NotePlayer in der Sprachreferenz.

Der NotePlayer ist – wie QuickTime – unter Linux nicht verfügbar.

SpriteSurface

Mit diesem Steuerelement kann man Animationen programmieren. Wird die Run-Methode von SpriteSurface aufgerufen, werden die Menüleiste, alle Fenster und der Schreibtisch versteckt. Das gestattet der Animation, den gesamten Bildschirm zu verwenden. Die Animation wird mit Hilfe sogenannter Sprites realisiert. Ein Sprite ist ein Objekt, das ein Bild enthält und das über den Bildschirm bewegt werden kann. Das SpriteSurface-Objekt kümmert sich automatisch um das Zeichnen des Hintergrundes und der Sprites und verschickt eine Nachricht, wenn zwei Sprites kollidieren. Außerdem bietet es dem Entwickler die Möglichkeit, auf Tastatureingaben des Anwenders zu reagieren.

Eine vollständige Liste der Eigenschaften von SpriteSurface finden Sie in der Sprachreferenz.

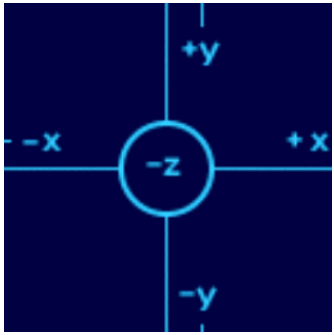
RB3DSpace

Das RB3DSpace-Steuerelement wird verwendet, um Animationen im dreidimensionalen Raum darzustellen. Es arbeitet in Verbindung mit den 3D-Klassen und bietet Funktionen zum Laden, Gruppieren und Manipulieren von 3D-Objekten. Das Basis-3D-Objekt ist von der Object3D-Klasse abgeleitet; 3D-Objekte werden dem Anwender über das RB3DSpace-Steuerelement präsentiert.

Das RB3DSpace-Steuerelement benötigt entweder die QuickTime3D- oder die Quesa-Bibliotheken. Wenn Sie Mac OS Classic verwenden, wird die Installation der QuickTime-Bibliotheken standardmäßig vorgenommen und Sie müssen keine gesonderte Installation durchführen. Unter Mac OS X müssen Sie die Quesa-Bibliotheken installieren, die Sie auf der CD oder auf der REALbasic-Website finden.

Um loszulegen, platzieren Sie ein RB3DSpace-Steuerelement in einem Fenster und aktivieren seine DebugCube-Eigenschaft. Wechseln Sie in die Laufzeitumgebung. Falls die 3D-Bibliotheken installiert sind, werden Sie den dreidimensionalen Raum mit den beschrifteten Koordinaten-Achsen sehen.

Abb. 113: Ein leerer 3D-Raum mit Koordinatensystem



Horizontale und vertikale Achse sind x- und y-Achse. Demzufolge ist die z- Achse die Nah-/Fern-Achse. Der „Ursprung“ oder der 0,0,0-Punkt befindet sich dort, wo die Achsen aufeinandertreffen. Wenn Sie mit angeschalteter Debug-Cube-Eigenschaft kein Koordinatensystem sehen, sind die benötigten Bibliotheken offenbar nicht auf Ihrem Rechner installiert.

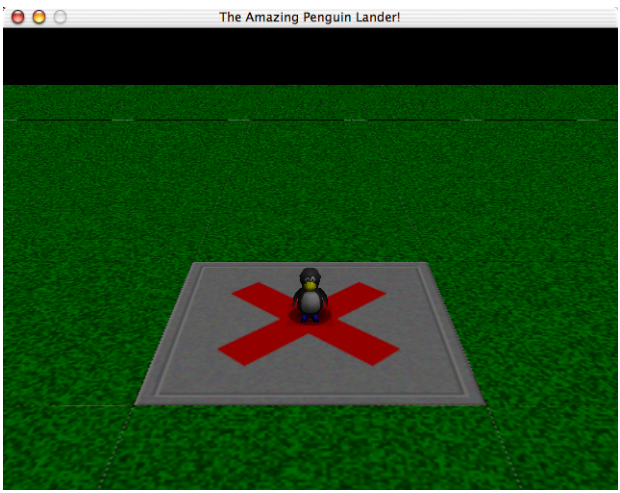
Nachdem Sie sichergestellt haben, dass die benötigten Bibliotheken vorhanden sind, können Sie ein Objekt in den 3D-Raum laden und die Kamera davon wegbewegen. Um einen Eindruck davon zu bekommen, wie man 3D-Objekte lädt und animiert, schauen Sie sich die Beispiele für RB3DSpace-Steuerelemente in der Sprachreferenz an.

Das RB3DSpace-Steuerelement wird am besten durch ein Beispiel illustriert. In folgender Abbildung wird die Position und der Flugpfad des Pinguins im 3D-Raum mit der Maus manipuliert.

- **Yaw**- oder Links-/Rechts-Bewegungen
- **Pitch**- oder Aufwärts-/Abwärts-Bewegungen
- **Roll**- oder Drehungen mit oder gegen den Uhrzeigersinn.

Außerdem können Sie die Position der „Kamera“ im 3D-Raum ändern.

Abb. 114: Ein Beispiel eines RB3DSpace Steuerelements



Weitere Informationen finden Sie in den Beispiel-Projekten auf der REALbasic-CD und in der Sprachreferenz.

Weitere Steuerelemente

PopupArrow Steuerelement

In REALbasic können über die Eigenschaft Facing sowohl die Ausrichtung als auch die Größe der Popup-Pfeile beeinflusst werden. Normalerweise wird ein PopupArrow-Steuerelement als ein Teil eines benutzerdefinierten Steuerelements verwendet. Die Ausrichtung des Pfeils zeigt an, ob das benutzerdefinierte Steuerelement zusätzliche Informationen oder Optionen anzeigen kann. Abbildung 115 zeigt die möglichen Ausrichtungen.

Abb. 115: Beispiele für das PopupArrow Steuerelement

DisclosureTriangle Steuerelement

Ein DisclosureTriangle-Steuerelement (Aufklappdreieck) wird verwendet, um hierarchische Listen anzuzeigen, wie sie auch in der Listendarstellung des Finders verwendet werden. Sie können die Ausrichtung (nach links oder rechts) und den Status (aufgeklappt oder nicht) beeinflussen.

Abb. 116: Aufklappdreiecke

LittleArrows Steuerelement

Das LittleArrows-Steuerelement kann verwendet werden, um dem Anwender eine Schnittstelle zum Blättern (z.B. in einer Liste) zur Verfügung zu stellen. Mit den zwei Methoden Up und Down können Sie feststellen, ob und welchen Pfeil der Anwender angeklickt hat.

Abb. 117: LittleArrows Steuerelement

ChasingArrows Steuerelement

Das ChasingArrows-Steuerelement (sich verfolgende, drehende Pfeile) geben dem Anwender eine visuelle Rückmeldung, dass gerade ein zeitaufwendiger Vorgang abläuft. Die ChasingArrows werden angezeigt, indem man ihre Visible-Eigenschaft auf True setzt.

Abb. 118: ChasingArrows Steuerelement

RBScript Steuerelement

Das RBScript-Steuerelement erlaubt es dem Anwender, REALbasic-Code innerhalb einer compilierten Anwendung zu schreiben und auszuführen. Solche Skripts werden in Maschinencode compiliert.

Sie übergeben den REALbasic-Code, den Sie ausführen lassen wollen, an die Source-Eigenschaft und führen ihn aus, indem Sie die Run-Methode aufrufen. Details zu den Funktionen, Kontrollstrukturen und Befehlen, die das RBScript-Steuerelement bietet, finden Sie in der Sprachereferenz.

Steuerelemente zur Kommunikation

REALbasic bietet Steuerelemente, mit denen Ihr Programm über den seriellen Port (z.B. mit einem Modem oder einem anderen Gerät) oder über ein Netzwerk mit anderen Geräten per TCP/IP kommunizieren kann.

Serial

Das Serial-Objekt wird zwar mit einem Icon angezeigt, wenn man es in der Entwicklungsumgebung auf ein Fenster zieht, es hat aber keine Interface-Elemente, die der Benutzer später sehen kann. Es wird nur dazu benutzt, den Code zur Kommunikation mit der seriellen Schnittstelle auszuführen. Weitere Informationen finden Sie bei der Beschreibung des Serial-Steuerelements in der Sprachreferenz.

Da das Serial-Objekt keine Unterklasse von Control ist, kann eine Serial-Instanz auch direkt im Programmcode erzeugt werden. Dadurch ist es möglich, auf die serielle Schnittstelle zuzugreifen, ohne dass man das Serial-Steuerelement auf einem Fenster platzieren muss.

TCPSocket

Das TCPSocket-Objekt wird zwar mit einem Icon angezeigt, wenn man es in der Entwicklungsumgebung auf ein Fenster zieht, es hat aber kein eigenes Interface-Element, das der Benutzer später sehen kann. Es wird nur dazu benutzt, um den Code zur Kommunikation mit anderen Computern im Intranet oder Internet auszuführen.

Da TCPSocket keine Unterklasse von Control ist, kann eine Socket-Instanz auch direkt im Programmcode erzeugt werden. Dadurch ist es möglich, auf TCP/IP-Verbindungen zuzugreifen, ohne dass man das Socket-Steuerelement auf einem Fenster platzieren muss. Weitere Informationen zum Socket-Steuerelement finden Sie in der Sprachreferenz.

Vor der Veröffentlichung von REALbasic 5 wurde diese Funktionalität von der Socket-Klasse übernommen. Die Socket-Klasse wurde durch die TCPSocket-Klasse ersetzt und mit der Einführung der SSLSocket-, UDPSocket- und ServerSocket-Klassen wurden die Kommunikationsmöglichkeiten weiter verbessert. Wenn in einer Ihrer bestehenden Anwendungen die Socket-Klasse verwendet wird, sollten Sie den Code auf die TCPSocket-Klasse umschreiben.

SSLSocket

Das SSLSocket-Steuerelement ist dem TCPSocket-Steuerelement sehr ähnlich. Mit SSLSocket integrieren Sie TCP/IP-Kommunikation mit SSL-Verschlüsselung. Es werden die SSL-Versionen 2 und 3 und TLS (Transport Layer Security) in der Version 1 unterstützt.

SSLSocket besitzt kein eigenes Icon in der Steuerelementepalette. Da es sich nicht von der Control-Klasse ableitet, können Sie ein SSLSocket-Element per Code erzeugen. Sie können es auch über das Kontextmenü eines Fensters zu einem Fenster hinzufügen. Um eine SSL-Verbindung über SSLSocket herzustellen, setzen Sie die Secure-Eigenschaft auf Wahr (TRUE) und verwenden die Connect-Methode.

SSLSocket ist nur in der Pro-Version von REALbasic verfügbar.

ServerSocket

Über die SocketServer-Klasse können Sie über einen Port mehrere TCP/IP-Verbindungen gleichzeitig einrichten. Wird eine Verbindung an diesem Port aufgenommen, leitet ServerSocket diese Verbindung an einen anderen Anschluss weiter und widmet sich wieder dem gleichen Port. ServerSocket besitzt die Fähigkeit, die Versorgung mit TCPSockets der Anzahl der Verbindungen anzupassen. Aufgrund der Verzögerung zwischen dem Eintreffen einer Verbindung, deren Weiterleitung, der Erzeugung einer neuen Socket und der Wiederaufnahme des Abhör-Prozesses, ist es ohne die ServerSocket-Klasse schwierig, eine gleichwertige Funktionalität zu gewährleisten. Im Fall, dass zwei Verbindungen nahezu zeitgleich aufgebaut werden, kann es passieren, dass eine Verbindung fallen gelassen wird, da keine freie Socket zur Verfügung stand. Für weitere Informationen über die ServerSocket-Klasse geben Sie das Stichwort „Server Socket“ in die elektronische Sprachreferenz ein.

UDPSocket

UDPSocket unterstützt die Kommunikation über eine UDP-Verbindung (User Datagram Protocol). Das UDPSocket-Element besitzt kein eigenes Icon in der Steuerelementepalette, kann aber, da es sich nicht von der Control-Klasse ableitet, per Code erzeugt oder über das Kontextmenü eines Fensters eingefügt werden.

UDP ist die Basis für Hochgeschwindigkeits-Netzwerkverkehr. Es ist ein verbindungsloses Protokoll mit einem sehr kleinen Overhead, ist aber nicht so sicher wie TCP. Da keine Verbindung aufgebaut werden muss, ist zur Verwendung von UDPSocket nicht annähernd so viel Aufwand zu treiben wie beim TCPSocket-Steuerelement.

UDP Sockets können in verschiedenen Modi eingesetzen werden. Am häufigsten werden Sie zum „Multicasting" eingesetzt. Multicasting ähnelt einem Chatroom: Sie können den Chatroom betreten und sich mit allen anderen, die sich im Chatroom aufhalten, unterhalten. Für weitere Informationen über die UDPSocket-Klasse geben Sie das Stichwort „UDPSocket" in die elektronische Sprachreferenz ein.

„Einfaches" Netzwerk

REALbasic enthält eine Version der UDP- und TCPSockets, die speziell für die Kommunikation mit anderen REALbasic Programmen entwickelt wurde. Diese Klassen heißen EasyUDPSocket und EasyTCPSocket. Zusätzlich gibt es noch die Klasse AutoDiscovery, die automatisch das Netzwerk durchsucht, um weitere Computer zu finden, die über die „Easy"-Sockets kommunizieren. Werden weitere Computer gefunden, ist es recht einfach, sich gegenseitig Nachrichten zukommen zu lassen.

Diese Klassen machen die Benutzung von Netzwerken recht einfach, sind aber nicht dafür ausgelegt, um mit anderen Applikationen, wie zum Beispiel einem HTTP-Server, zu kommunizieren.

Weitere Informationen über die „Easy"-Klassen finden Sie im Abschnitt „Netzwerke ganz einfach" auf Seite 376 und in der Sprachreferenz.

Das Toolbar-Objekt und das Standard Toolbar-Objekt

Mit Hilfe des Toolbar- und des Standard Toolbar-Objekts können Sie in Ihren Anwendungen Toolbars (Werkzeugleisten) erstellen. Das Toolbar-Objekt können Sie mit Standard-Symbolen von Apple besetzen oder eigene Icons verwenden. Weitere Informationen siehe Sprachreferenz.

Das Timer-Objekt

Das Timer-Objekt führt Programmcode in vorbestimmten Intervallen aus. Es wird zwar mit einem Icon angezeigt, wenn man es in der Entwicklungsumgebung auf ein Fenster zieht, hat aber keine für den Benutzer sichtbaren Interface-Elemente. Siehe „Timer-Steuerelement" in der Sprachreferenz.

Steuerelemente für das Arbeiten mit Datenbanken

Es gibt zwei Steuerelemente für spezielle Zwecke, die nur in Projekten relevant sind, in denen auf Datenbanken zugegriffen wird: Das DatabaseQuery- und das DataControl-Steuerelement. Ausführliche Informationen zu diesen Steuerelementen finden Sie im Kapitel „Datenbankprogrammierung mit REALbasic" auf Seite 334.

DatabaseQuery

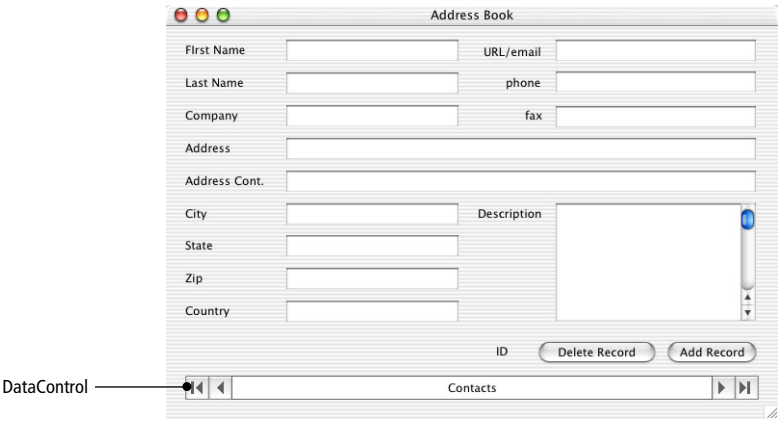
Das DatabaseQuery-Steuerelement kann verwendet werden, um SQL-Abfragen an eine Datenbank zu richten. Dies kann auch über die REALbasic-Sprache erfolgen (ohne dass Sie das DatabaseQuery-Steuerelement überhaupt verwenden müssen). Ein DatabaseQuery-Steuerelement, das Sie einem Fenster hinzugefügt haben, ist in der späteren Anwendung nicht sichtbar.

Zur Verwendung des DatabaseQuery-Steuerelements schreiben Sie einen SQL-Ausdruck und weisen diesen der SQLQuery-Eigenschaft des Steuerelements zu.
Das SQLQuery wird automatisch ausgeführt, wenn das Fenster geöffnet wird, auf dem sich das DatabaseQuery-Steuerelement befindet. Dieses besitzt außerdem die Methode RunQuery, mit der die Abfrage ausgeführt werden kann.
Sehen Sie sich hierzu das Beispiel im Abschnitt „DatabaseQuery und Objektverknüpfungen“ auf Seite 342 an.

DataControl

Das DataControl-Steuerelement ist ein „zusammengesetztes“ Steuerelement, das es ermöglicht, auf einfache Weise ein Datenbank-Frontend zu bauen. Es besteht aus vier Navigations-Knöpfen (erster Datensatz, voriger Datensatz, nächster Datensatz und letzter Datensatz) und der Beschriftung.

Abb. 119: Ein DataControl-Steuerelement auf der Eingabemaske einer Adressdatenbank.



Wenn das DataControl mit einer Datenbank und Steuerelementen, die Daten anzeigen, verknüpft wird, können Sie ein vollständig funktionierendes Datenbankfrontend ohne weitere Programmierung erzeugen. Ein Beispiel hierzu finden Sie im Abschnitt „Das DataControl Steuerelement“ auf Seite 344.

ActiveX Steuerelemente

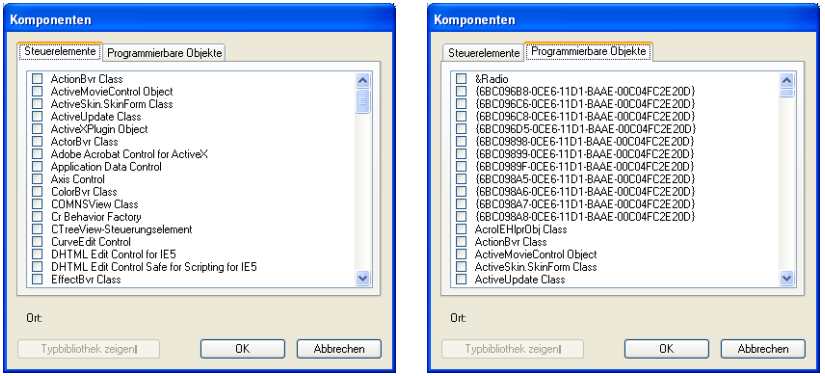
In der Windows-Version von REALbasic können Sie ActiveX-Steuerelemente in die Steuerelementepalette aufnehmen. Ein ActiveX-Steuerelement erscheint unterhalb der Standard-Steuerelemente, verhält es sich wie ein integraler Bestandteil von REALbasic und steht Ihnen für Ihre Anwendungen zur Verfügung.

Abb. 120: Steuerelementepalette mit ActiveX-Steuerelementen

Um unter Windows ein ActiveX-Steuerelement auf der Steuerelementepalette zu platzieren, unternehmen Sie folgendes:

1. Wählen Sie **Datei/ActiveX Komponenten.** Daraufhin erscheint der ActiveX-Komponenten-Dialog, der alle auf Ihrem Computer installierten ActiveX-Steuerelemente und -Objekte anzeigt. Der Dialog enthält die Karteikarten „Steuerelemente“ und „Programmierbare Objekte“.

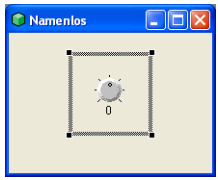
Abb. 121: ActiveX-Komponenten



2. Wählen Sie die Steuerelemente oder programmierbaren Objekte aus, die Sie der Steuerelementepalette hinzufügen wollen, und klicken Sie anschließend auf **OK**. Steuerelemente besitzen in der Regel eigene Icons. Programmierbare Objekte verwenden das OLEContainer-Icon. Die so hinzugefügten ActiveX-Komponenten werden Teil des aktuellen Projekts, nicht aber von REALbasic selbst. Um diese ActiveX-Komponenten für ein anderes Projekt zu verwenden, müssen Sie diese Schritte wiederholen.

   Ein ActiveX-Steuerelement lässt sich wie jedes andere Steuerelement auf einem Fenster ablegen. Draggen Sie es auf ein Fenster und verwenden Sie den Code-Editor, um Code für das ActiveX-Steuerelement zu schreiben. Für genauere Information zur Programmierung von ActiveX-Komponenten ziehen Sie bitte die Dokumentation von Microsoft zu Rate. Diese finden Sie in der MSDN-Bibliothek unter: http://msdn.microsoft.com/library/.

Abb. 122: Ein ActiveX-Regler-Steuerelement



OLEContainer Steuerelement

Mit dem OLEContainer-Steuerelement können Sie ActiveX-Steuerelemente in Ihre Benutzeroberfläche einbetten. ActiveX gibt es nur unter Windows.

Object Binding – Objektverknüpfung

Sobald Sie die Steuerelemente Ihrer Applikation auf einem Fenster platziert haben, können Sie ihnen im Code-Editor die gewünschte Funktion geben. Beispielsweise können Sie festlegen, wie sich ein PushButton verhalten soll, wenn er angeklickt wird, indem Sie im Action-Event des PushButtons den entsprechenden Programmcode ablegen.

In einigen Fällen können Sie Funktionen realisieren, ohne dass Sie dafür Programmzeilen tippen müssen. Dafür verantwortlich ist „Object Binding“, die Objektverknüpfung.

Dank Objektverknüpfung wird eine bestimmte Aktion ausgeführt, wenn sich eine Eigenschaft eines Steuerelements verändert, ohne dass dafür Programmcode vorhanden sein muss. Ein einfaches Beispiel für Objektverknüpfung haben wir bereits im Abschnitt über das MoviePlayer-Steuerelement auf Seite 127 präsentiert. Die zwei PushButtons „Play“ und „Stop“ wurden mit dem MoviePlayer-Steuerelement verknüpft. Diese Verknüpfungen spezifizierten die Funktion der PushButtons, es musste nirgendwo Code dafür geschrieben werden.

Beim MoviePlayer-Beispiel handelt es sich um eine gerichtete Verknüpfung: Das Steuerelement, mit dem der Anwender interagiert, ist das Quellobjekt. Das Steuerelement, von dem aufgrund der Objektverbindung eine Aktion ausgeführt wird, ist das Zielobjekt.

Um eine Objektverknüpfung herzustellen, gehen Sie folgendermaßen vor:

1. Drücken Sie die Shift- und Befehlstaste (Windows: Strg- und Shift-Taste) und ziehen Sie die Maus vom Quell- zum Ziel-Steuerelement (dabei erscheint eine Linie zwischen beiden Steuerelementen).
2. Es öffnet sich der Dialog „Objektverknüpfung“ (Object Binding) mit einer Liste der verfügbaren Aktionen. Die in REALbasic fest eingebauten Objektverknüpfungen finden Sie in Tabelle 1. REALbasic kann um benutzerdefinierte Objektverknüpfungen erweitert werden.
3. Wählen Sie die gewünschte Aktion aus und klicken Sie auf **OK**.

Im Fenster erscheint eine Linie, die die beiden Steuerelemente verbindet. Wenn Sie Ihr Programm ausführen, arbeitet die Verknüpfung so, als ob Sie für die Aktion entsprechenden Programmcode eingegeben hätten. Die Objektverknüpfungen im MoviePlayer-Beispiel sind äquivalent zu den Programmzeilen `MoviePlayer1.Play` und `MoviePlayer1.Stop`, die man in den Action-Events des Play- und Stop-Knopfes hätte aufrufen können.

Falls Sie einmal vergessen haben, welche Verknüpfung durch eine Linie repräsentiert wird, können Sie die Linie selektieren. Im Eigenschaftenfenster erscheint dann die gewählte Aktion. Um eine Verknüpfung zu modifizieren, müssen Sie die bestehende Verknüpfung zunächst löschen (indem Sie die Linie selektieren und Backspace drücken). Jetzt können Sie die Verknüpfung mit der gewünschten Aktion neu anlegen.

Die folgende Tabelle zeigt die in REALbasic fest eingebauten Verknüpfungen:

Tabelle 1. In REALbasic integrierte Objektverknüpfungen

Quellobjekt	Zielobjekt	Verknüpfung
PushButton oder BevelButton	*MoviePlayer*	Film im MoviePlayer abspielen, wenn Knopf angeklickt wird. Film im MoviePlayer anhalten, wenn Knopf angeklickt wird.
PushButton oder BevelButton	*DataBaseQuery*	DatabaseQuery ausführen, wenn Knopf angeklickt wird.
PushButton oder BevelButton	*ListBox*	Aktiviere Knopf, wenn in der ListBox etwas selektiert wurde.
RadioButton	*DatabaseQuery*	DatabaseQuery ausführen, wenn RadioButton selektiert wird. DatabaseQuery ausführen, wenn RadioButton deselektiert wird.
PushButton	*DataControl*	Fügt Datensatz ein, wenn der PushButton gedrückt wird. Aktualisiert Datensatz, wenn der PushButton gedrückt wird. Löscht Datensatz, wenn der PushButton gedrückt wird. Neuer Datensatz, wenn der PushButton gedrückt wird.
BevelButton	*DataControl*	Fügt Datensatz ein, wenn der BevelButton gedrückt wird. Aktualisiert Datensatz, wenn der BevelButton gedrückt wird. Löscht Datensatz, wenn der BevelButton gedrückt wird. Neuer Datensatz, wenn der BevelButton gedrückt wird.
RadioButton	*DataControl*	Fügt Datensatz ein, wenn der RadioButton selektiert wird. Aktualisiert Datensatz, wenn der RadioButton selektiert wird. Löscht Datensatz, wenn der RadioButton selektiert wird. Neuer Datensatz, wenn der RadioButton selektiert wird. Fügt Datensatz ein, wenn der RadioButton deselektiert wird. Aktualisiert Datensatz, wenn der RadioButton deselektiert wird. Löscht Datensatz, wenn der RadioButton deselektiert wird. Neuer Datensatz, wenn der RadioButton deselektiert wird.
CheckBox	*DataControl*	Fügt Datensatz ein, wenn die CheckBox angewählt wird. Aktualisiert Datensatz, wenn die CheckBox angewählt wird. Löscht Datensatz, wenn die CheckBox angewählt wird. Neuer Datensatz, wenn die CheckBox angewählt wird. Fügt Datensatz ein, wenn die CheckBox abgewählt wird. Aktualisiert Datensatz, wenn die CheckBox abgewählt wird. Löscht Datensatz, wenn die CheckBox abgewählt wird. Neuer Datensatz, wenn die CheckBox abgewählt wird.
RadioButton	*ListBox*	Aktiviere RadioButton, wenn in der ListBox etwas selektiert wurde.
RadioButton	*MoviePlayer*	MoviePlayer-Film abspielen, wenn RadioButton selektiert ist. MoviePlayer-Film stoppen, wenn RadioButton selektiert ist. MoviePlayer-Film abspielen, wenn RadioButton deselektiert ist. MoviePlayer-Film stoppen, wenn RadioButton deselektiert ist.
DatabaseQuery	*PopupMenu*	PopupMenu mit dem Ergebnis von DatabaseQuery verknüpfen. Parameter von DatabaseQuery mit PopupMenu verknüpfen. Parameter von DatabaseQuery mit der RowTag-Methode von PopupMenu verknüpfen. (Beispiele für diese Verknüpfungen finden Sie im Abschnitt „DatabaseQuery und Objektverknüpfungen“ auf Seite 342.)

Tabelle 1. In REALbasic integrierte Objektverknüpfungen

Quellobjekt	Zielobjekt	Verknüpfung
DatabaseQuery	*ListBox*	Verknüpfe Listbox mit dem Ergebnis von DatabaseQuery. Verknüpfe die Parameter von DatabaseQuery mit ListBox. DatabaseQuery ausführen, wenn ListBox den Fokus erhält. DatabaseQuery ausführen, wenn ListBox den Fokus verliert.
CheckBox	*NotePlayer*	NotePlayer aktivieren, wenn Checkbox mit einem Häkchen versehen ist.
Checkbox	*DatabaseQuery*	DatabaseQuery ausführen, wenn Checkbox mit einem Häkchen versehen ist. DatabaseQuery ausführen, wenn Checkbox deselektiert ist.
Checkbox	*Contextual Menu*	ContextualMenu aktivieren, wenn Checkbox mit einem Häkchen versehen ist.
CheckBox	*PopupMenu*	PopupMenu aktivieren, wenn Checkbox mit einem Häkchen versehen ist.
CheckBox	*PopupArrow*	PopupArrow aktivieren, wenn Checkbox mit einem Häkchen versehen ist.
CheckBox	*DisclosureTriangle*	DisclosureTriangle aktivieren, wenn Checkbox mit einem Häkchen versehen ist.
CheckBox	*SpriteSurface*	SpriteSurface aktivieren, wenn Checkbox mit einem Häkchen versehen ist.
CheckBox	*ImageWell*	ImageWell aktivieren, wenn Checkbox mit einem Häkchen versehen ist.
Checkbox	*ListBox*	Aktiviere ListBox, wenn Checkbox mit Häkchen versehen ist. Aktiviere Checkbox, wenn ein ListBox-Eintrag ausgewählt ist.
CheckBox	*TabPanel*	TabPanel aktivieren, wenn CheckBox mit einem Häkchen versehen ist.
CheckBox	*ChasingArrows*	ChasingArrows aktivieren, wenn CheckBox mit einem Häkchen versehen ist.
CheckBox	*EditField*	Aktiviere EditField, wenn CheckBox mit Häkchen versehen ist.
CheckBox	*Canvas*	Aktiviere Canvas, wenn CheckBox mit Häkchen versehen ist.
CheckBox	*Progressbar*	Aktiviere EditField, wenn CheckBox mit Häkchen versehen ist.
Checkbox	*MoviePlayer*	MoviePlayer-Film abspielen, wenn Checkbox mit Häkchen versehen ist. MoviePlayer-Film stoppen, wenn Checkbox mit Häkchen versehen ist. MoviePlayer-Film abspielen, wenn Checkbox nicht selektiert ist. MoviePlayer-Film stoppen, wenn Checkbox nicht selektiert ist. Aktiviere MoviePlayer, wenn Checkbox mit einem Häkchen versehen ist.
CheckBox	*Slider*	Aktiviere Slider, wenn CheckBox aktiviert.
CheckBox	*ScrollBar*	Aktiviere ScrollBar, wenn CheckBox aktiviert.
ListBox	*LittleArrows*	LittleArrows aktivieren, wenn in ListBox ein Eintrag ausgewählt ist.
ListBox	*DisclosureTriangle*	DisclosureTriangle aktivieren, wenn in ListBox ein Eintrag ausgewählt ist.
ListBox	*PopupMenu*	PopupMenu aktivieren, wenn in ListBox ein Eintrag ausgewählt ist.
ListBox	*PopupArrow*	PopupArrow aktivieren, wenn in ListBox ein Eintrag ausgewählt ist.
ListBox	*ChasingArrows*	ChasingArrows aktivieren, wenn in ListBox ein Eintrag ausgewählt ist.
ListBox	*ContextualMenu*	ContextualMenu aktivieren, wenn in ListBox ein Eintrag ausgewählt ist.
ListBox	*ScrollBar*	ScrollBar aktivieren, wenn in ListBox ein Eintrag ausgewählt ist.
ListBox	*NotePlayer*	NotePlayer aktivieren, wenn in ListBox ein Eintrag ausgewählt ist.
ListBox	*Slider*	Slider aktivieren, wenn in ListBox ein Eintrag ausgewählt ist.
ListBox	*MoviePlayer*	Spielt MoviePlayer-Film, wenn die ListBox den Fokus bekommt. Stoppt MoviePlayer-Film, wenn die ListBox den Fokus bekommt. Spielt MoviePlayer-Film, wenn die ListBox den Fokus verliert. Stoppt MoviePlayer-Film, wenn die ListBox den Fokus verliert. Aktiviere MoviePlayer, wenn in der ListBox ein Eintrag gewählt wird.
ListBox	*ImageWell*	ImageWell aktivieren, wenn in ListBox ein Eintrag ausgewählt ist.
ListBox	*Canvas*	Canvas aktivieren, wenn in ListBox ein Eintrag ausgewählt ist.

Tabelle 1. In REALbasic integrierte Objektverknüpfungen

Quellobjekt	Zielobjekt	Verknüpfung
ListBox	*EditField*	Verknüpfe EditField mit den String-Daten der ListBox (kopiert selektiertes Objekt in das EditField). Aktiviere EditField, wenn in ListBox ein Eintrag selektiert ist.
ListBox	*ProgressBar*	ProgressBar aktivieren, wenn in ListBox ein Eintrag ausgewählt ist.
ListBox	*TabPanel*	TabPanel aktivieren, wenn in ListBox ein Eintrag ausgewählt ist.
EditField	*PopupMenu*	Verknüpfe EditField mit PopupMenu (der ausgewählte Menüeintrag wird in das EditField übernommen). Verknüpfe EditField mit der RowTag-Methode von PopupMenu.

Es können außerdem benutzerdefinierte Objektverknüpfungen programmiert werden. Mehr dazu im Abschnitt „Benutzerdefinierte Objektverknüpfungen“ auf Seite 328.

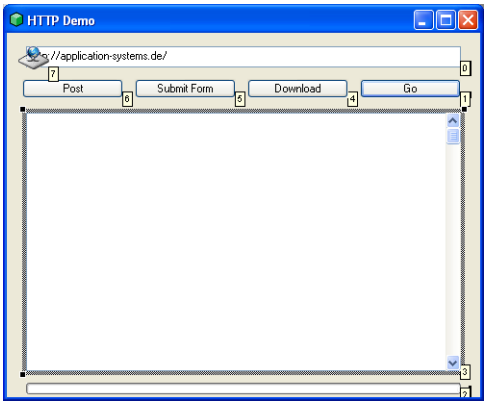
Ändern der Steuerelementereihenfolge (Tab-Reihenfolge)

Die Reihenfolge, in der die einzelnen Steuerelemente den Fokus zur Eingabe von der Tastatur erhalten (EditFields und ListBoxes), nennt man Control-Order oder auch Tab-Reihenfolge. Sie wird durch Control-Layers festgelegt, also Ebenen, in denen die Steuerelemente liegen. Das muss man sich so vorstellen, dass jedes Steuerelement auf einer eigenen Ebene liegt und diese Ebenen übereinander gestapelt sind. Wird ein Fenster geöffnet, erhält das Objekt, das am weitesten unten liegt und zur Eingabe geeignet ist, den Fokus. Sie können die Objekte auf drei verschieden Arten durch die Ebenen bewegen:

- Verwenden Sie das **Format**-Menü, um die Steuerelemente durch die Ebenen zu bewegen.
- Benutzen Sie die Dialogbox **Reihenfolge**.
- Verwenden Sie das Eigenschaftenfenster.

Das **Format**-Menü enthält Menübefehle zum Ändern der Reihenfolge der Steuerelemente. Zuerst können Sie sich über **Format/Zeige Steuerelemente-Reihenfolge** die aktuelle Reihenfolge der Steuerelemente anzeigen lassen. Dadurch wird an jedem Steuerelement eine Nummer eingeblendet.

Abb. 123: Reihenfolge der Steuerelemente



Das Steuerelement mit der höchsten Nummer enthält den Fokus als erstes. Zum Ändern der Reihenfolge wählen Sie ein Steuerelement aus und rufen einen der folgenden Befehle aus dem **Format**-Menü auf:

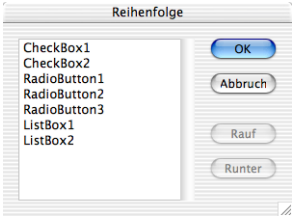
- Nach vorne bewegen

- An den Anfang setzen
- Nach hinten bewegen
- Ans Ende setzen

Dann werden die entsprechenden Markierungen der Steuerelemente auf den neuesten Stand gebracht und die neue aktuelle Position in der Reihenfolge der Steuerelemente wird angezeigt.

Die Reihenfolge der Steuerelemente kann auch mit Hilfe der Dialogbox **Reihenfolge** verändert werden.

Abb. 124: Die Dialogbox Reihenfolge

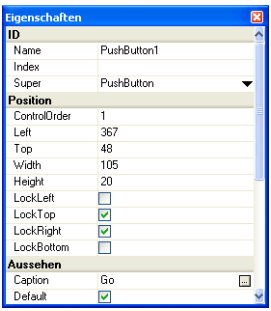


So ändern Sie die Reihenfolge mit Hilfe der Dialogbox:

1. Wählen Sie **Format/Reihenfolge**.
2. Wählen Sie das Objekt in der Liste, dessen Tab-Position Sie ändern möchten.
3. Bewegen Sie dieses mit den Knöpfen **Rauf** und **Runter**, um die Reihenfolge zu verändern.

Schließlich können Sie auch im Eigenschaftenfenster unter **Position** die Position eines Steuerelements in der Steuerelementereihenfolge ändern. Sie können diesen Wert durch Eingabe eines neuen Wertes und anschließendes Drücken von Return modifizieren.

Abb. 125: Die ControlOrder-Eigenschaft eines Steuerelements im Eigenschaftenfenster



Wenn Sie **Format/Zeige Steuerelemente-Reihenfolge** aufrufen, während Sie im Eigenschaftenfenster die Reihenfolge der Steuerelemente ändern, spiegeln sich die Änderungen erst wieder, nachdem Sie im Eigenschaftenfenster einen neuen Wert abgespeichert haben.

Ausrichten der Steuerelemente

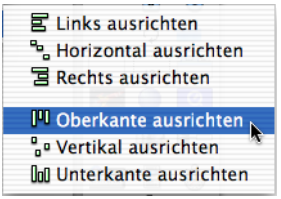
Der Interface Assistant in REALbasic erleichtert es sehr, die Positionen der einzelnen Steuerelemente an anderen Elementen auszurichten. Mit einer gepunkteten Hilfslinie und einem Raster sehen Sie sehr schnell, ob die Elemente auf gleicher Höhe liegen.

Hinweis: Ist der Interface Assistant mal im Weg, dann können Sie ihn vorübergehend ausblenden, indem Sie die Befehlstaste drücken, während Sie ein Element bewegen.

Zur Ausrichtung der Steuerelemente geht man so vor:

1. Klicken Sie auf das Steuerelement, dessen Position bereits korrekt ist.
2. Wählen Sie **Format/Ans Ende setzen**, um zu gewährleisten, dass das Steuerelement an seinem Platz bleibt, wenn die anderen Elemente bewegt werden.
3. Wählen Sie mit gedrückter Shift-Taste nacheinander alle Elemente, die Sie gemeinsam ausrichten wollen.
4. Je nachdem, in welcher Richtung Sie die Objekte anordnen wollen, wählen Sie unter dem Menüpunkt **Objekte ausrichten** die entsprechende Option.

Abb. 126: Das Untermenü „Objekte ausrichten"



Äquidistantes Ausrichten

Um Objekte mit einem gleichmäßigen Abstand zu versehen, gehen Sie folgendermaßen vor:

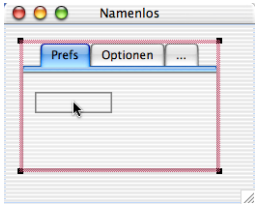
1. Klicken Sie auf ein Steuerelement, um es auszuwählen.
2. Während Sie die Shift-Taste drücken, wählen Sie nacheinander mindestens zwei weitere Elemente.
3. Wählen Sie unter **Objekte ausrichten** die Option **Horizontal ausrichten** oder **Vertikal ausrichten**.

Die Steuerelemente-Hierarchie

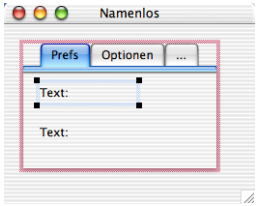
Manchmal erzeugen Sie Teile Ihrer Benutzeroberfläche, indem Sie Steuerelemente in einem anderen Steuerelement platzieren. So können Sie z.B. das GroupBox-Steuerelement dazu verwenden, um andere Steuerelemente auszurichten, in der Regel RadioButtons, Checkboxen, Popup Menüs und EditFields. Auch das TabPanel und das PagePanel sind dafür ausgelegt, andere Steuerelemente zu beherbergen. Die Steuerelemente-Hierarchie der IDE ermöglicht es Ihnen, mit Gruppen von Steuerelementen zu arbeiten.

Ein Steuerelement, das andere Steuerelemente umfasst, wird Eltern-Steuerelement genannt. Alle Steuerelemente, die sich vollständig innerhalb des Eltern-Steuerelements befinden, heißen Kinder-Steuerelemente. Diese Zuordnung erfolgt automatisch, wenn Sie die Steuerelemente in folgender Reihenfolge erzeugen: Eltern – Kinder. Damit ist gemeint, dass Sie zuerst das Steuerelement erzeugen, das die anderen umschließt und anschließend die Kinder-Steuerelemente, oder Sie duplizieren bestehende Kinder-Steuerelemente und platzieren sie innerhalb der Grenzen des Eltern-Steuerelements.

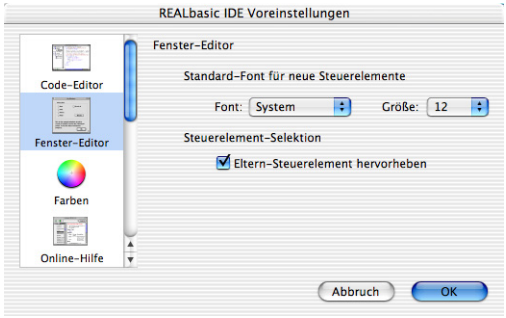
Wenn Sie auf diese Art und Weise ein Kinder-Steuerelement hinzufügen, erhält das Eltern-Steuerelement eine Umrahmung. Diese dient als visuelle Bestätigung, dass Sie gerade ein Kinder-Steuerelement in einem Eltern-Steuerelement platzieren. Die folgende Abbildung zeigt, wie ein StaticText-Steuerelement als Kind in einem TabPanel platziert wird.



Das Eltern-Steuerelement wird auch immer dann umrahmt, wenn Sie eines seiner „Kinder" mit der Maus anwählen. In der folgenden Abbildung klickt der Benutzer beispielsweise auf ein StaticText-Steuerelement. Das Eltern-Steuerelement (TabPanel) wird dabei ebenfalls umrahmt.



Über eine Option in den Voreinstellungen des Fenstereditors können Sie dieses Verhalten ein- oder ausschalten. Wählen Sie dazu **REALbasic/Einstellungen** unter Mac OS X (oder **Bearbeiten/Einstellungen** unter Mac OS Classic oder Windows). Wählen Sie dann die Dialogseite **Fenstereditor** aus und deaktivieren bzw. aktivieren Sie **Eltern-Steuerelement hervorheben**.



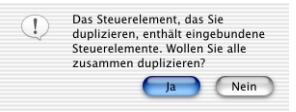
Auf der Dialogseite **Farben** des Voreinstellungen-Dialogs können Sie unter **Eltern hervorheben** eine Farbe für die Umrahmung festlegen.

Wenn ein Steuerelement nicht vollständig von einem anderen Steuerelement umschlossen ist, dann ist dieses nicht automatisch ein Kind-Steuerelement. Es überlappt lediglich das andere Steuerelement.

Wenn Sie ein Kind-Steuerelement aus einem Eltern-Steuerelement herausbewegen, dann ist dieses Steuerelement nicht länger ein Kind. Wenn Sie das Steuerelement vollständig innerhalb eines anderen Steuerelements positionieren, wird es zum Kind-Steuerelement dieses Steuerelements.

Ein Eltern-Steuerelement kann einschließlich seiner Kinder-Steuerelemente kopiert werden. REALbasic fragt dann nach, ob die Kinder-Steuerelemente mit kopiert werden sollen.

Abb. 127: Nachfrage beim Kopieren eines Steuerelements mit zugeordneten Kinder-Steuerelementen

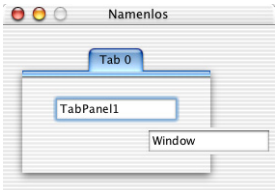


Steuerelemente können über mehrere Ebenen ineinander verschachtelt sein. So können Sie z.B. eine GroupBox innerhalb eines TabPanels platzieren und anschließend diverse RadioButtons innerhalb der GroupBox anlegen. In diesem Fall ist die GroupBox das Eltern-Steuerelement der RadioButtons und das TabPanel ist das Eltern-Steuerelement der GroupBox. Um diese Hierarchie automatisch anzulegen, müssen Sie die Steuerelemente in der Reihenfolge ihrer Hierarchie erstellen: Zuerst das TabPanel, dann die GroupBox und schließlich die RadioButtons.

Eigenschaften der Steuerelement-Hierarchie

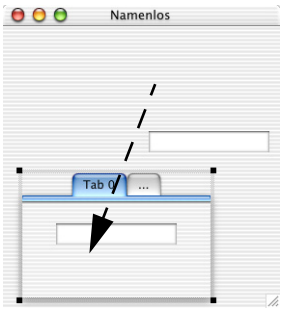
Wenn Sie ein Kinder-Steuerelement duplizieren und das Duplikat im Elternobjekt belassen, ist es automatisch ein Kind. Wenn Sie das Duplikat jedoch aus dem Elternobjekt heraus verschieben, ist es kein Kind mehr. Ein Steuerelement muss sich komplett innerhalb des Elternobjekts befinden, damit es als ein Kind anerkannt wird.

Im folgenden Beispiel zeigen beide EditFields die Identität ihrer Eltern an. Das untere wurde aus dem oberen dupliziert, wurde jedoch aus dem TabPanel heraus verschoben. Es ist kein Kind mehr.



Wenn Sie das untere EditField wieder in das TabPanel zurückbewegen, dann ist es wieder ein Kind des TabPanels.

Das Verschieben eines Eltern-Steuerelements bewegt seine Kinder ebenso. Im folgenden Beispiel wurde das oben gezeigte TabPanel nach unten verschoben. Es nimmt das obere EditField mit sich, lässt das untere aber zurück.



- Das Löschen eines Eltern-Steuerelements löscht alle Kinder-Steuerelemente.

- Durch Verstecken eines Eltern-Steuerelements werden auch alle Kinder unsichtbar. Der Wert der **Visible**-Eigenschaft der Kinder bleibt dabei erhalten.
- Das Anzeigen eines Elternobjekts zeigt auch alle Kinder an, deren **Visible**-Eigenschaft gesetzt ist.
- Das Deaktivieren des Elternobjekts deaktiviert auch alle Kinder, der Wert der Enabled-Eigenschaft der Kinder bleibt aber erhalten. In der IDE werden durch Deaktivieren eines Containers auch alle Kinder-Steuerelemente deaktiviert, die Enabled-Eigenschaft wird dabei nicht aktualisiert.

Hier sehen Sie eine deaktivierte GroupBox in der IDE mit zwei RadioButtons und zwei EditFields als Kind-Objekte. Wenn die GroupBox deaktiviert ist, sind alle vier Kinder deaktiviert. Da alle Kinder deaktiviert sind, wenn das Elternobjekt deaktiviert ist, kann man in der Applikation die EditFields nicht ansteuern.

Abb. 128: Eine deaktivierte GroupBox, die Ihre Kindobjekte ebenfalls deaktiviert.

Das Aktivieren eines Eltern-Steuerelements aktiviert nur die Kinder-Steuerelemente, deren Enabled-Eigenschaft gesetzt ist. Im folgenden Beispiel ist nur bei den Kinder-Steuerelementen der unteren Reihe die Enabled-Eigenschaft gesetzt. Obwohl die GroupBox aktiviert ist, bleiben also die Steuerelemente der oberen Reihe deaktiviert.

Abb. 129: Ein aktiviertes GroupBox-Steuerelement

Wenn dieses Verhalten für ein Steuerelement unerwünscht ist, können Sie die Steuerelemente-Hierarchie durchbrechen, indem Sie im Open-Event des Fensters die **Parent**-Eigenschaft des Steuerelements auf Nil setzen. Dann wird sich das Steuerelement unabhängig von seinem früheren Eltern-Steuerelement verhalten.

Definieren von Menüs und Menü-Einträgen

Der in REALbasic eingebaute Menüeditor hilft Ihnen beim Anlegen von Menüleisten, Menüs und Menüeinträgen für ein Programm.

Menüleisten hinzufügen

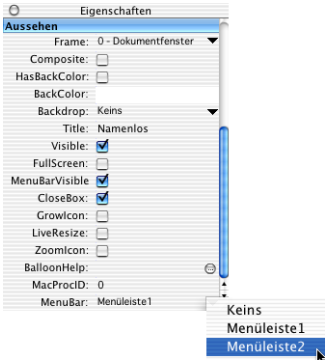
Wenn Sie das erste Mal ein Desktop-Applikations-Projekt anlegen, zeigt das Projektfenster ein Fenster und eine Menüleiste mit dem Namen Menüleiste1 an. Das ist die Standard-Menüleiste, die automatisch für Ihre Applikation verwendet wird. Sie können weitere Menüleisten zu Ihrem Projekt hinzufügen und diese einem Fenster zuordnen. Auf dem Macintosh ist es so, dass die Menüleiste des Fensters, das aktiviert wird, die aktuelle Menüleiste ersetzt. Unter Linux und Windows besitzt jedes Fenster seine eigene Menüleiste.

Bei Windows-MDI-Applikationen wird die globale Menüleiste als Menüleiste für den MDI-Rahmen verwendet. Dokumentfenster einer MDI-Applikation dürfen keine Menüleiste haben. Schwimmende Fenster dagegen dürfen eine Menüleiste besitzen.

Um zu Ihrem Projekt eine zusätzliche Menüleiste hinzuzufügen, unternehmen Sie folgendes:

1. Bringen Sie das Projektfenster in den Vordergrund, indem Sie darauf klicken. Sollte es von anderen Fenstern verdeckt sein, wählen Sie **Fenster/Projekt** (Mac: ⌘-0 bzw. Windows: Strg-0)
2. Wählen Sie **Ablage/Neue Menüleiste**. Eine neue Menüleiste erscheint im Projektfenster.
3. Folgen Sie den Schritten im Abschnitt „Menüs hinzufügen“ auf Seite 144, um die Menüleiste mit Menüs und Menüeinträgen zu versehen.
4. Um einem Fenster eine bestimmte Menüleiste zuzuordnen, öffnen Sie dessen Eigenschaftenfenster. Wählen Sie dann im Popup-Menü der MenuBar-Eigenschaft die gewünschte Menüleiste aus.

Abb. 130: Zuordnen einer Menüleiste zu einem Fenster



Dieses Feature ist besonders für Windows-Anwendungen interessant, da jedes Fenster seine eigene Menüleiste haben kann. Wenn Sie dieses Feature auf dem Mac einsetzen, ändert sich die globale Menüleiste am oberen Bildschirmrand immer dann, wenn ein Fenster mit einer anderen Menüleiste in den Vordergrund rückt.

5. Um die globale Menüleiste zu ändern, setzen Sie die MenuBar-Eigenschaft der Application-Klasse auf die gewünschte Menüleiste.

   Beim Anlegen eines neuen Projekts wird diesem automatisch eine neue App-Klasse hinzugefügt. Dabei handelt es sich um eine Unterklasse der Application-Klasse. Um die globale Menüleiste zu ändern, müssen Sie lediglich der MenuBar-Eigenschaft der App-Klasse eine andere Menüleiste zuweisen.

Abb. 131: Ändern der globalen Menüleiste

Bei MDI-Anwendungen unter Windows ist die globale Menüleiste die des MDI-Rahmens.

Menüs hinzufügen

Die Vorgehensweise zum Einrichten eines Menüs und seiner Einträge gliedert sich in 4 Schritte. Diese Schritte sind sowohl für die voreingestellte globale Menüleiste als auch für jede neue Menüleiste, die Sie Ihrem Projekt hinzufügen, immer die gleichen.

- Das Menü mit dem Menüeditor anlegen.
- Die Menüeinträge mit dem Menüeditor hinzufügen.

- Den Menüeintrag aktivieren. Ein inaktiver Menüeintrag ist deaktiviert und kann nicht selektiert werden.
- Einen Menü-Handler anlegen. Ein Menü-Handler ist eine Methode, mit der festgelegt wird, wie REALbasic auf das Auswählen eines (aktivierten) Menüeintrags durch den Benutzer reagiert. Ein aktiver Menüeintrag ohne Menü-Handler kann zwar ausgewählt werden, es wird aber keine Aktion ausgelöst.

  Es gibt zwei Möglichkeiten, einen Menüpunkt zu aktivieren. Soll der Menüeintrag durchweg aktiv sein, können Sie seine **AutoEnable**-Eigenschaft im Eigenschaftenfenster einschalten. Sobald Sie einen Menü-Handler hinzufügen, ist der Menüeintrag funktional. Sie könnten zum Beispiel für den Menüeintrag **Neu** die AutoEnable-Eigenschaft verwenden, mit dem sich ein neues Dokument in der Anwendung erzeugen ließe.

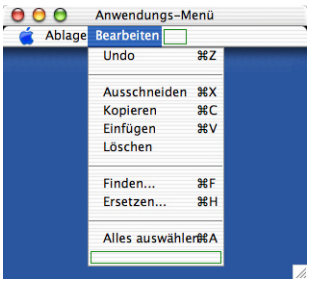
  Soll der Menüeintrag nicht immer aktiv sein, müssen Sie REALbasic mitteilen, wann es den Menüeintrag zu aktivieren hat. Es könnte zum Beispiel sein, dass Sie einen Menüeintrag zum Exportieren verwenden, der jedoch nur aktiv sein soll, wenn der Benutzer auch ein passendes Objekt zum Exportieren ausgewählt hat. Deaktivieren Sie in diesem Fall die AutoEnable-Eigenschaft des Menüeintrags. Dann ist der Menüeintrag solange inaktiv, bis der Benutzer die Menüzeile ansteuert. Zu diesem Zeitpunkt können Sie prüfen, ob die Bedingungen zum Aktivieren des Menüeintrags erfüllt sind. Bei einem Menüeintrag **Save** müsste man zum Beispiel überprüfen, ob das Dokument bereits gespeichert wurde oder ob seit dem letzten Speichern Änderungen vorgenommen wurden.

Ab REALbasic Version 5 können Menüeinträge auch per Programmcode hinzugefügt und entfernt werden. Weitere Informationen dazu finden Sie in der Sprachreferenz unter dem Stichwort „MenuItem Klasse“.

REALbasic legt in der deutschen Version automatisch die Menüs Ablage und Bearbeiten an, während die englische Version die Menüs File und Edit anlegt. Jedes Programm sollte zumindest über ein **Ablage**-Menü mit dem Menüpunkt **Beenden** verfügen. Das Bearbeiten-Menü können Sie nur entfernen, wenn Ihr Programm keine Steuerelemente hat, die mittels der Menüpunkte im Bearbeiten-Menü verändert werden könnten. Ein neues Menü entsteht so:

1. Klicken Sie das Projektfenster an, um es in den Vordergrund zu holen. Wenn es von anderen Fenstern verdeckt sein sollte, klicken Sie auf **Fenster/Projekt**.
2. Mit einem Doppelklick auf das Menü-Objekt öffnen Sie den Menüeditor.

Abb. 132: Der Menüeditor



3. Klicken Sie auf das gepunktete Rechteck in der Menüleiste des Menüeditors, um es zu selektieren.
4. Geben Sie im Eigenschaftenfenster den Namen des Menüs ein. Der Text wird daraufhin in der Menüleiste erscheinen.

Tabelle 2. Eigenschaften des Menu-Objekts

Name	Beschreibung
Super	Die Klasse, von der das Menu-Objekt abgeleitet ist.
Name	Der interne Name des Menu, der dazu verwendet wird, es im Programmcode anzusprechen.
Text	Der Text, der in der Menüleiste erscheinen soll.

Abb. 133: Eigenschaftenfenster eines Menüeintrags

Ein Hilfe-Menü anlegen

Die meisten Mac-Anwendungen haben ein Hilfe-Menü, das sich immer ganz rechts in der Menüleiste befindet. Das mindeste, was sich in diesem Menü befinden sollte, sind die Menüpunkte **Über Erklärungen** und **Erklärungen ein**. Das Hilfe-Menü kann außerdem Menüpunkte umfassen, die dem Benutzer Zugriff auf „Mac Hilfe"-Dateien oder ein speziell für die Anwendung geschriebenes Hilfe-System gestatten. Ein Hilfemenü, das bereits die nötigen Menüpunkte für die Erklärungen enthält, erzeugen Sie so:

1. Erzeugen Sie einen neuen Menüpunkt am Ende Ihrer Menüleiste.
2. Setzen Sie die Text-Eigenschaft auf **Help**.

Das Menü wird beim Start des Programms auf einem deutschen System automatisch „Hilfe" heißen und die beiden Menüpunkte „Über Erklärungen" und „Erklärungen ein" enthalten. Im Menüeditor erscheinen diese jedoch nicht.

Hinzufügen von Menüpunkten

Menüpunkte können im Menüeditor ganz leicht in ein Menü eingefügt werden. Ebenso lassen sich Tastaturkürzel definieren. Denken Sie daran, dass die Tastenkürzel von links nach rechts in den Menüs zugeordnet werden. Wenn Sie dieselbe Taste zweimal verwenden, wird der weiter rechts gelegene Menüpunkt nicht damit angesprochen. Ferner sind einige Kürzel reserviert. Nach den Apple Macintosh Human Interface Guidelines sind dies:

Tabelle 3. Reservierte Tastenkürzel

Menü	Tasten	Befehl
Ablage (File)	⌘-N	Neu (New)
Ablage (File)	⌘-O	Öffnen… (Open…)
Ablage (File)	⌘-W	Schließen (Close)
Ablage (File)	⌘-S	Sichern (Save)
Ablage (File)	⌘-P	Drucken… (Print…)
Ablage (File)	⌘-Q	Beenden (Quit)
Bearbeiten (Edit)	⌘-Z	Undo
Bearbeiten (Edit)	⌘-X	Ausschneiden (Cut)
Bearbeiten (Edit)	⌘-C	Kopieren (Copy)
Bearbeiten (Edit)	⌘-V	Einfügen (Paste)
Bearbeiten (Edit)	⌘-A	Alles auswählen (Select All)
Bearbeiten (Edit)	⌘-Punkt	Abbruch eines laufenden Vorgangs
Bearbeiten (Edit)	⌘-M	Minimieren (Minimize)

Einen neuen Menüpunkt erzeugen Sie so:

1. Selektieren Sie im Menüeditor das Menü, in das der Menüpunkt eingefügt werden soll.

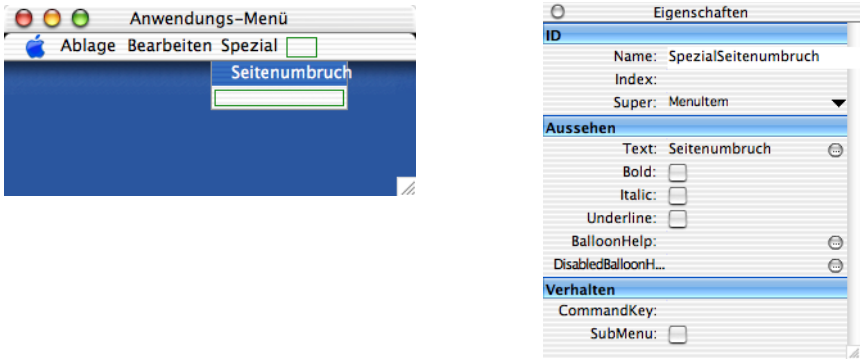
2. Klicken Sie auf das gepunktete Rechteck am Ende des Menüs, um es zu selektieren.
3. Im Eigenschaftenfenster geben Sie schließlich den Text für den Menüpunkt ein. Der Menüeditor bietet einen Default-Namen an.
4. Fügen Sie, falls gewünscht, einen Tastatur-Kurzbefehl hinzu, indem Sie der CommandKey-Eigenschaft einen Buchstaben zuweisen.
5. Optional: Selektieren Sie die Eigenschaft Bold, Italic oder Underline oder fügen Sie einen Hilfe-Text hinzu.

Hinweis: Auch wenn der Menüeditor kleine Buchstaben für Tastaturkürzel gestattet, sollten dennoch große Buchstaben verwendet werden.

Wenn Sie die alt-Taste im Tastaturkürzel verwenden möchten, fügen Sie „option“ bei der Command-Key-Eigenschaft hinzu. „option-F“ legt z.B. die Tastenkombination ⌘-alt-F als Tastatur-Kurzbefehl fest.

Folgende Abbildung zeigt einen Seitenumbruch-Menüeintrag, der dem Spezial-Menü hinzugefügt wurde.

Abb. 134: Ein neuer Menüeintrag im Spezial-Menü



Das Mac OS X-Programm-Menü

Mac OS X-Programme haben standardmäßig ein Menü links vom Ablage-Menü. Es erhält automatisch den Namen der Applikation. Wenn Sie Ihre Applikation in der Entwicklungsumgebung von REALbasic testen, heisst der Menüpunkt **REALbasic**. Wenn Sie ein eigenständig lauffähiges Mac OS X-Programm erzeugen, bekommt das Menü den Namen, den Sie im Dialog „Applikation erzeugen“ bei „Mac OS-Programmeinstellungen“ eingegeben haben.

Der Menüeintrag „Beenden”

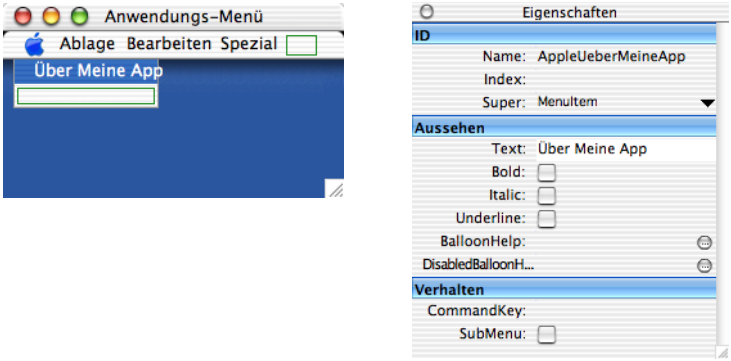
Der Menüeintrag „Beenden“, der bei Mac OS Classic-Anwendungen im Ablage-Menü erscheint, befindet sich unter Mac OS X am Ende des Applikations-Menüs. Automatisch hinzugefügt werden Menüeinträge für das Ein- und Ausblenden der aktuellen Applikation und ein „Alle einblenden“ Menüeintrag.

Im Menüeditor wird der Beenden-Menüeintrag immer im Ablage-Menü angezeigt. Wenn das Programm unter Mac OS X gestartet wird, wandert er automatisch ins Applikations-Menü. Das Ablage-Menü bleibt unter Mac OS X so lange deaktiviert, bis Sie ihm einen Menüeintrag hinzufügen.

Dem Applikations-Menü fügen Sie einen Menüeintrag hinzu, indem Sie diesen im Apfel-Menü des Menüeditors unterbringen. Wenn die Applikation für Mac OS Classic compiliert wird, erscheinen diese Menüpunkte im Apfel-Menü.

In folgender Abbildung wird ein „Über...“-Menüeintrag dem Applikations-Menü in Mac OS X und dem Apfel-Menü im Classic Mac OS hinzugefügt.

Abb. 135: Hinzufügen eines „Über Meine App"-Menüpunkts ins Applikationsmenü (Mac OS X) bzw. ins Apfel-Menü (Mac OS Classic).



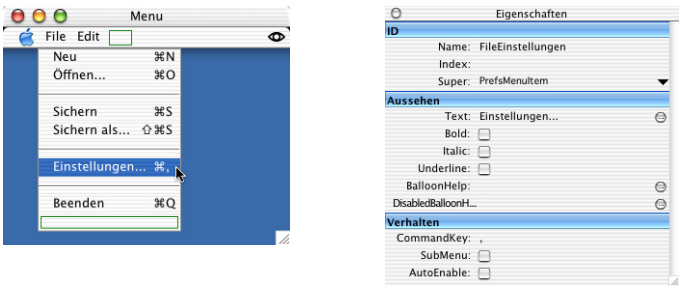
Hinzufügen eines „Einstellungen"-Menüeintrags

Unter Mac OS X befindet sich der „Einstellungen"-Menüpunkt standardmäßig im Menü der Applikation. Unter anderen Betriebssystemen erscheint er aber im „Ablage-", „Bearbeiten-" oder „Datei-" Menü.

Wie kann man nun einen Menüpunkt an zwei Stellen gleichzeitig unterbringen? Dies ist recht einfach. Erstellen Sie den „Einstellungen"-Menüeintrag in dem Menü, das Sie für Ihre Windows- und Mac OS Classic-Version verwenden wollen.

Im Eigenschaftenfenster des Menüeintrags stellen Sie den Wert der **SuperClass**-Eigenschaft von **MenuItem** auf **PrefsMenuItem**. Jeder Menüpunkt, der von **PrefsMenuItem** abgeleitet ist, wird unter Mac OS X automatisch unter dem Applikations-Menü einsortiert (oder unter dem **REALbasic**-Menü, wenn die Applikation in der IDE läuft), verbleibt unter Windows und Classic Mac OS aber an der Stelle, an der Sie ihn eingesetzt haben. Der **Einstellungen**-Menüpunkt in der folgenden Abbildung erscheint unter Windows und unter Mac OS Classic im **Datei**-Menü, unter Mac OS X aber im Applikations-Menü.

Abb. 136: Einer Applikation den Menüpunkt „Einstellungen" hinzufügen



Wenn Sie außerdem einen „Über dieses Programm"-Menüpunkt hinzugefügt haben, erscheint dieser als erster Menüpunkt im Programm-Menü, gefolgt vom **Einstellungen**-Menüpunkt.

Gemäß der Apple-Richtlinien für die Mac OS X-Benutzeroberfläche sollten Sie „⌘ ," (⌘-Komma) als Tastaturkürzel für den Einstellungen-Menüpunkt verwenden.

Untermenüs anlegen

Untermenüs sind Menüpunkte, bei deren Auswahl ein Menü nach rechts ausklappt. Der Menüpunkt selbst kann nicht ausgewählt werden. Er dient nur als Titel für das Untermenü.

Ein Untermenü wird wie folgt angelegt:

1. Klicken Sie im Menüeditor auf den Menüpunkt, der das Untermenü beherbergen soll.

2. Im Eigenschaftenfenster kreuzen Sie die Eigenschaft „Submenu“ an. Ein neues Untermenü-Objekt erscheint im Menüeditor.
3. Im Menüeditor klicken Sie auf das gepunktete Rechteck, das rechts neben dem Menüpunkt neu aufgetaucht ist.
4. Im Eigenschaftenfenster geben Sie den Namen für den Untermenüpunkt ein.

Untermenüs bieten dem Benutzer einen schnellen Zugriff auf eine Gruppe von Menüpunkten. Allerdings können sie einen Computer-Neuling verwirren, da sie den direkten Blick auf einzelne Menüpunkte versperren. Durchsucht ein Benutzer die Menüs nach einem bestimmten Menüpunkt, sieht er sich möglicherweise die Untermenüs nicht an. Überlegen Sie sich genau, an welche Zielgruppe Sie sich mit Ihrem Programm wenden, bevor Sie Untermenüs einsetzen. Sollten es in der Mehrzahl ungeübte Benutzer sein, dann sollten Sie die Wahlmöglichkeiten eher in einer Dialogbox unterbringen als in einem Untermenü.

Untermenüpunkte können selbst auch wieder ein Untermenü enthalten. Aber auch hier gilt, dass damit Funktionen des Programms versteckt werden und es möglicherweise zu kompliziert wird, in verschachtelten Menüs nach dem richtigen Kommando zu suchen.

Verschieben von Menüs und Menüpunkten

Ein Menüpunkt kann durch Ziehen mit der Maus an eine andere Position im Menü verschoben werden. Menüpunkte können aber nur innerhalb desselben Menüs verschoben werden. Wenn der Menüpunkt in ein anderes Menü wandern soll, dann müssen Sie ihn löschen und in dem anderen Menü neu eingeben.

Sie können auch Menüs innerhalb der Menüzeile verschieben. Draggen Sie das Menü in der Menüleiste nach links oder rechts, ein vertikaler Balken zeigt die Position des Menüs an.

Löschen eines Menüpunkts

Einen Menüpunkt können Sie löschen, indem Sie ihn anklicken und die Backspace-Taste drücken.

Einfügen einer Trennlinie

Man verwendet Trennlinien, um Menüpunkte optisch zu Gruppen zusammenzufassen. Um eine Trennlinie einzufügen, legt man einen Menüpunkt an und tippt in die Text-Eigenschaft einfach ein Minus („-“).

Arrays aus Menü-Einträgen

In manchen Fällen können Sie die Menüpunkte nicht im Voraus bestimmen, da sich diese je nach Kontext oder Umgebung ändern, in der das Programm läuft. Ein Beispiel hierfür ist das Font-Menü, das normalerweise in jedem Programm vorhanden ist, das Text mit Stilinformationen unterstützt. Der Programmierer weiß vorher nicht, welche Fonts auf dem Rechner des Anwenders installiert sind.

Erzeugen Sie das Menü. Erzeugen Sie einen Menü-Eintrag, geben ihm einen Namen und setzen seine Index-Eigenschaft auf 0, um ein Array aus Menüeinträgen zu erzeugen. Dann müssen Sie den Code schreiben, der das Array mit den Namen der Fonts füllt, die auf dem Rechner des Anwenders installiert sind. Vorausgesetzt, dass keine Fonts installiert oder entfernt werden, so lange das Programm läuft, reicht es, die Font-Liste einmal beim Programmstart aufzubauen.

Platzieren Sie dazu den Code in den Open-Event-Handler der Application-Klasse. Dieser wird ausgeführt, wenn das Programm gestartet wird. Das *App*-Objekt, das automatisch für Ihr Projekt erzeugt wird, ist der richtige Ort dafür.

Folgender Code füllt das Font-Menü, wenn das Programm gestartet wird. Es verwendet die eingebaute Font-Funktion, die den Namen des i.ten Fonts auf dem Rechner des Anwenders liefert.

```
Dim i, nFonts as Integer
nFonts=FontCount-1
```

```
//baue Font-Menü auf
FontFontName(0).text=Font(0)
For i=1 to nFonts
 m=New FontFontName //erzeugt neuen Menü-Eintrag
 m.Text=Font(i)//liefert Namen des i.ten Fonts
Next
```

Apple Macintosh User Interface Guidelines

Wenn Sie ein Programm nicht für den Eigenbedarf, sondern für andere Anwender entwickeln, sollten Sie sich unbedingt ausreichend Zeit für das Design der Benutzeroberfläche nehmen. Diese entscheidet darüber, wie gern und wie effektiv man Ihr Programm einsetzen wird. Falls ein Anwender nicht binnen 15 Minuten ein Erfolgserlebnis mit Ihrer Software hat, wird er sich frustriert abwenden und versuchen, sein Problem anderweitig zu lösen.

Ein Programm wirkt um so professioneller, je durchdachter seine Benutzeroberfläche ist. Dabei ist es für den Anwender sehr hilfreich, wenn er für die Benutzung Ihres Programms auf Fertigkeiten zurückgreifen kann, die er bei der Verwendung anderer Produkte erlangt hat.

Der REALbasic Interface Assistant™ hilft Ihnen, eine professionelle Benutzeroberfläche zu gestalten. Wenn Sie ein Steuerelement in die Nähe eines Fensterrandes draggen, zeigt Ihnen REALbasic eine Hilfslinie an, die Ihnen vermittelt, wie weit Sie das Steuerelemente draggen dürfen. Diese Linie erscheint 20 Pixel vor dem rechten, linken und unteren Rand des Fensters und 14 Pixel vor dem oberen Rand. Wenn Sie ein Steuerelement auf ein anderes Steuerelement zubewegen, zeigt REALbasic eine Linie im Abstand von 12 Pixeln (der empfohlene Abstand zwischen zwei Steuerelementen) an.

Es ist etwas völlig anderes, ein Programm zu benutzen, als die Benutzeroberfläche eines Programms zu gestalten. Wenn Sie es noch nicht getan haben, sollten Sie sich einmal die *Macintosh Human Interface Guidelines* ansehen. Hier wird genau beschrieben, warum die Macintosh-Benutzeroberfläche so realisiert wurde, wie wir sie kennen. Wenn Sie sich an diese Vorgaben halten, wird Ihr Programm für jeden mit dem Mac vertrauten Anwender einfach zu benutzen sein.

Die *Macintosh Human Interface Guidelines* finden Sie im Internet unter www.devworld.apple.com.

Kapitel 4 Programmierung in BASIC

In diesem Kapitel lernen Sie, wie BASIC aufgebaut ist und welche Unterschiede es zwischen REALbasic und BASIC gibt.

Inhalt

- Datentypen
- Ablegen von Werten in Eigenschaften und Variablen und Konstanten
- Ausführen von Instruktionen in Methoden
- Mehrfaches Ausführen von Instruktionen in Schleifen
- Verzweigen

BASIC versus REALbasic

BASIC wurde um 1960 erfunden, um Leuten das Programmieren beizubringen. Alles, was Programmieren zu kompliziert macht, wurde aus BASIC entfernt. Das hatte zur Folge, dass BASIC lange Zeit als ungeeignet für komplexe Programme galt, was allerdings weniger an der Sprache selbst lag, als vielmehr an den Implementierungen, die existierten.

Frühere Implementierungen von BASIC verwendeten einen Interpreter, der jede Instruktion des Programms las und ausführte. Das bedeutet, dass jedesmal, wenn das Programm lief, Zeile für Zeile übersetzt und ausgeführt wurde, so wie es ein Simultandolmetscher im wirklichen Leben tun würde. Für andere Sprachen existierten bereits Compiler, die das Programm in Maschinencode übersetzen und abspeichern. Das hat den Vorteil, dass solche Programme zur Ausführung nicht mehr übersetzt werden müssen. Das macht die Sache natürlich schon erheblich schneller.

REALbasic hat einen eingebauten Compiler. Das bedeutet, dass der von REALbasic erzeugte Code so schnell wie nur möglich ausgeführt wird. REALbasic ist eine moderne objektorientierte Sprache. Wenn Sie erst mit dem Programmieren anfangen, dann sagt Ihnen das jetzt nicht viel, aber Sie werden schon noch herausfinden, was das bedeutet. REALbasic verbindet die einfache Benutzbarkeit von BASIC mit modernen Programmierkonzepten und erlaubt so, auf einfache Weise sehr leistungsfähige Programme zu schreiben. Dabei ist es nicht nötig, im einzelnen zu wissen, wie das Betriebssystem genau funktioniert. Daher ist es einfacher, ein Programm zu schreiben und darüber hinaus kann man auch Programme schreiben, die unter Umständen auf ganz anderen Betriebssystemen funktionieren, da sie nichts enthalten, was nur auf einem bestimmten Betriebssystem funktioniert.

Ablegen von Werten in Eigenschaften und Variablen

Sollen Informationen auf einem Computer permanent gespeichert werden, dann sichert man sie in einem Dokument auf der Festplatte. In einem Programm ist es oft nötig, Informationen nur vorübergehend im Speicher abzulegen. Sie können sich das als Ansammlung von Schubladen vorstellen. Jede dieser Schubladen ist irgendwo im Speicher und um sie voneinander unterscheiden zu können und den Inhalt wiederzufinden, bekommt jede Schublade einen Namen, was schon besser ist als eine Speicheradresse, denn das wäre nur eine Zahl. Je nachdem, wie man diese Speicherplätze nutzt, nennt man sie Variablen oder Eigenschaften (Properties).

Was sind Eigenschaften?

Variablen, die Werte speichern, die ein Objekt – zum Beispiel ein Fenster – beschreiben, nennt man Eigenschaften (Properties). Der Titel im Titelbalken des Fensters, seine Breite oder seine Höhe werden in Eigenschaften des Fensters

gespeichert. Wenn ein Fenster geöffnet wird, dann werden diese Eigenschaften in den Speicher kopiert. Sie können sie über ihren Namen erreichen. Dadurch können Sie die Daten, die in den Eigenschaften abgelegt sind, auslesen oder andere Werte hineinschreiben. Wollen Sie beispielsweise, dass sich der Titel eines Fensters ändert, wenn der Benutzer einen Knopf anklickt, schreiben Sie in die Eigenschaft „Title“ des Fensters einen anderen Wert. Jede Eigenschaft kann eine bestimmte Art von Daten enthalten. Manche speichern Text, so wie die Eigenschaft „Title“ eines Fensters, andere speichern Zahlen, so wie beispielsweise die Eigenschaft „Width“, die die Breite des Fensters angibt.

Variablen

Manchmal kommt es vor, dass Sie einen Wert speichern wollen, der nicht direkt zu einem Objekt gehört. Dafür benutzen Sie Variablen. Eine Variable ist so etwas wie eine Eigenschaft, nur dass sie keinem Objekt zugeordnet ist.

Datentypen

Um leistungsstarke Kommandos verwenden zu können und Zeit bei der Ausführung des Programmcodes zu sparen, muss ein Computer bestimmte Annahmen über die Beschaffenheit der Daten, die man verwendet, machen. Er muss wissen, ob es sich bei der Information, die er vorgesetzt bekommt, um eine Zahl, ein Datum, einen Text etc. handelt. Ohne eine Information über den Datentyp könnte 1 plus 1 im Ergebnis 2 sein, wenn es sich um eine Zahl handelt oder auch „11“ als String (Zeichenkette), wenn es sich um Zeichen handelt, die man so hintereinander hängt. REALbasic versteht sehr viele Datentypen, aber sechs sind ganz besonders wichtig: String, Integer, Single, Double, Boolean und Color.

String

Ein String ist eine Reihe von Buchstaben, also eine Zeichenkette. Jede Information lässt sich im Prinzip als String speichern. “Jeanette”, “12.06.1968”, “45.90” können Strings sein. Sie denken jetzt vielleicht „Moment mal, die letzten beiden sind doch ein Datum und eine Zahl, aber keine Zeichenkette“, aber sie sind es schon, wenn man sie im Programmtext in Anführungszeichen setzt. Die maximale Länge eines Strings ist nur vom Speicher begrenzt, der zur Verfügung steht.

Zwei Strings können mit dem „+“-Zeichen zu einem verbunden werden. “Big”+”Dog” ergibt also “BigDog”. Das war dann aber schon die gesamte „Mathematik“, die Strings ermöglichen. Alles andere, was man mit Strings anstellen kann, wird mit anderen Funktionen gemacht.

Integer

Ein Integer ist eine ganze Zahl zwischen etwa -2 Milliarden und +2 Milliarden. In anderen Programmiersprachen würde man dies als Long Integer bezeichnen. Weil es sich um Zahlen handelt, kann man mit Integer-Werten rechnen. Sie werden nicht in Anführungszeichen gesetzt und belegen 4 Bytes im Speicher.

Single

Ein Single ist eine Zahl, die auch Dezimalstellen haben darf. Es gibt keine Beschränkung des Wertes. In anderen Sprachen wäre dies eine Single Precision Real Number. Mit Singles kann man rechnen und sie belegen 4 Bytes im Speicher.

Double

Ein Double ist eine Zahl, die auch Dezimalstellen haben darf. Es gibt keine Beschränkung des Wertes. In anderen Sprachen wäre dies eine Double Precision Real Number. Mit Doubles kann man rechnen und sie belegen 8 Bytes im Speicher. Da Doubles mehr Dezimalstellen verwenden, um eine Zahl zu repräsentieren, sollten Sie diese verwenden, wenn die zusätzliche Genauigkeit für Ihre Berechnungen wichtig ist. Berechnungen mit Singles sind jedoch sowohl auf dem Macintosh als auch unter Windows schneller als mit Doubles.

Datentyp	Kleinster Wert	Größter Wert
Integer	-2147483648	2147483647

Datentyp	Kleinster Wert	Größter Wert
Single	1.175494 e-38	3.402823 e+38
Double	2.2250738585072013 e-308	1.7976931348623157 e+308

Boolean

Ein Boolean kann entweder den Wert „True“ (wahr) oder „False“ (falsch) annehmen. Boolean-Variablen enthalten zunächst den Wert „False“, können aber mit der True-Funktion auf den Wert „True“ gesetzt werden. Einige Objekte haben Eigenschaften vom Typ Boolean. Beispielsweise haben viele Objekte die Eigenschaft „Enabled“, die ein boole'scher Wert ist.

Color

Ein Color speichert den Wert einer REALbasic-Farbe. Ein Color-Wert besteht aus drei Komponenten und kann unter Verwendung dreier Farbmodelle eingestellt werden (RGB, HSV oder CMY). Da jede Komponente als ein Byte gespeichert wird, kann sie einen Wert zwischen 0-255 annehmen.

Mit folgenden Funktionen kann ein Color-Wert gesetzt werden.

Funktion	Farbmodell	Datentyp und Wertebereich der Parameter
RGB	Red-Green-Blue (Rot-Grün-Blau)	Integer (0-255)
HSV	Hue-Saturation-Value (Farbton-Sättigung-Helligkeit)	Double (0-1)
CMY	Cyan-Magenta-Yellow (Cyan-Magenta-Gelb)	Double (0-1)

Folgender Programmcode weist der fillcolor-Eigenschaft eines Rechtecks eine Farbe zu:

```
rectangle1.fillcolor=CMY(.35,.9,.6)
```

Ein RGB-Farbwert kann auch so zugewiesen werden:

```
&cRRGGBB
```

RRGGBB geben dabei jeweils in Hexadezimaldarstellung die rote, grüne und blaue Komponente an. Jede Komponente kann also einen Wert von 00 bis FF annehmen.

Beispiel: `rectangle.fillColor=&cFF594A`

Die drei Color-Funktionen können gegeneinander ausgetauscht werden. Nachdem Sie eine Farbe zugewiesen haben, können Sie jede Ihrer neun Eigenschaften auslesen. Folgendes Beispiel liefert den Red-Farbanteil und kann verwendet werden, auch wenn zum Zuweisen der Farbe das CMY-Modell verwendet wurde:

```
Dim redValue as integer
redValue=rectangle1.fillcolor.red
```

Farben können auch in der Farbpalette definiert werden. Klicken Sie auf ein freies Feld der Farbpalette, um die Farbauswahl anzuzeigen und eine Farbe auszuwählen. Die in der Farbpalette abgelegten Farben können Sie bequem per Drag & Drop zuweisen: Ziehen Sie einfach eine Farbe auf eine beliebige Eigenschaft im Eigenschaftenfenster, die einen Farbwert erwartet. Es ist außerdem möglich, eine Farbe aus der Farbpalette in den Code-Editor zu ziehen. In diesem Fall wird der RGB-Wert im „&c“-Format an der Cursorposition eingefügt.

Variant

Ein Variant ist ein spezieller Datentyp, der den Wert jedes beliebigen Datentyps, inklusive Objekten, speichern kann. Die Methode des Variant-Typs liefert einen Integer-Wert, anhand dessen man den Datentyp des enthaltenen Wertes bestimmen kann.

Von Type gelieferter Wert	Beschreibung
0	Nil
2	Integer
5	Double oder Single
8	String
9	Object
11	Boolean
16	Color

Abhängig vom Datentyp des Wertes, den der Variant enthält, können Sie eine der folgenden Eigenschaften dazu verwenden, den Wert in einen anderen Datentyp umzuwandeln.

Eigenschaft	Beschreibung
BooleanValue	Liefert den Wert des Variant als ein Boolean.
ColorValue	Liefert den Wert des Variant als ein Color.
DoubleValue	Liefert den Wert des Variant als ein Double.
IntegerValue	Liefert den Wert des Variant als ein Integer.
ObjectValue	Liefert den Wert des Variant als ein Object.
StringValue	Liefert den Wert des Variant als ein String.

Auch wenn Sie Datum- und Datum-/Uhrzeit-Werte in einem Date-Objekt speichern, ist dies kein Datentyp. Tatsächlich ist Date eine Klasse. Lesen Sie hierzu auch den Abschnitt „Objekte deklarieren“ auf Seite 157.

Ändern eines Datentyps

Es kommt oft vor, dass man einen Wert von einem Datentyp in einen anderen umwandeln muss. Das ist normalerweise deswegen so, weil man den Wert mit etwas verknüpfen möchte, das von einer Funktion kommt, die einen anderen Datentyp verwendet. Beispielsweise kann es sein, dass die Anzahl der Fenster gezählt wird, was naturgemäß mit einer Zahl geschieht, die Nummer des Fensters aber im Titel angezeigt werden soll, was in der Title-Eigenschaft geschehen muss, die aber ihrerseits eine Zeichenkette ist. Daher erhalten Sie eine Fehlermeldung, wenn Sie versuchen, dem Fenstertitel die Zahl zuzuweisen. Die Fehlermeldung weist darauf hin, dass die beiden Datentypen unterschiedlich, sprich nicht kompatibel sind. Daher muss man die Zahl zuerst in einen String umwandeln, bevor man sie zuweisen kann.

Für diesen Zweck gibt es die Str-Funktion (Str steht für String), die eine Zahl in einen String wandeln kann. Siehe „Str-Funktion“ in der Sprachreferenz. Umgekehrt kann man aus einem String eine Zahl machen, indem man die Val-Funktion verwendet (Val steht für Value (Wert)). Siehe „Val-Funktion“ in der Sprachreferenz.

Zuweisung für Eigenschaften

Die Syntax für solche Zuweisungen ist im allgemeinen:

```
objectName.propertyName=value
```

Haben Sie einen PushButton, der pushbutton1 heißt und Sie möchten, dass die Beschriftung „OK“ sein soll, dann müssen Sie der Caption-Eigenschaft den String „OK“ zuweisen. Das geht so:

```
pushbutton1.caption="OK"
```

Dies können Sie lesen wie „ändere den Beschriftungstext für pushbutton1 auf OK“. Diese Syntax gilt, wenn Sie eine Eigenschaft eines Steuerelements im gleichen Fenster ändern möchten. Wenn ein Steuerelement in einem Fenster eine Eigenschaft eines Objektes in einem anderen Fenster ändern soll, dann müssen Sie den Namen des Fensters (nicht den Inhalt seiner Title-Eigenschaft, das sind zwei verschiedene Dinge!) mit einbeziehen. Angenommen, es werden zwei Fenster benutzt und das eine heißt window1, das andere window2. Nun soll ein PushButton, der in window1 liegt, die Beschriftung von pushbutton1 in window2 ändern. Die Syntax lautet in diesem Fall:

```
window2.pushbutton1.caption="OK"
```

Würden Sie den Namen des Fensters window2 nicht angeben, dann müsste REALbasic annehmen, dass Sie ein Objekt namens pushbutton1 in window1 meinen, da dies das Fenster ist, in dem das Objekt liegt, das den Programmcode enthält, der gerade ausgeführt wird. Wenn Sie allerdings mehr als eine Kopie des Fensters (window2) verwenden würden, das das zu ändernde Objekt enthält, dann funktioniert diese Methode nicht. Wie Sie in solchen Fällen vorgehen können, kommt später noch.

Soll eine Eigenschaft des Fensters selbst geändert werden, muss der Name des Fensters nicht angegeben werden. Wenn ein PushButton den Namen des Fensters ändern soll, in dem er sich selbst befindet, dann ist folgende Zuweisung nötig:

```
title="Hello World"
```

In manchen Fällen sind Eigenschaften hierarchisch gruppiert. Das beste Beispiel hierfür ist das Canvas-Steuerelement. Dieses ist mit einem Set von Zeichenwerkzeugen ausgestattet, das es erlaubt, das Aussehen frei zu gestalten. Alle Zeichenwerkzeuge sind über die Graphics-Eigenschaft erreichbar. Folgende Zeile z.B. zeichnet ein Rechteck der Größe 100*50 an den Koordination 0,0 von Canvas 1:

```
Canvas1.Graphics.DrawRect 0,0,100,50
```

DrawRect ist eine Methode, die zur Graphics-Klasse gehört und nicht zur Canvas-Klasse. Die Graphics-Eigenschaft der Canvas-Klasse erlaubt es Ihnen jedoch, auf diese zuzugreifen.

Me und Self

REALbasic kennt zwei sehr nützliche Befehle, *Me* und *Self*. *Me* verweist auf das Steuerelement, mit dem Sie arbeiten. Wenn Sie z.B. die Caption-Eigenschaft von PushButton1 in einem der Event-Handler dieses Steuerelements auf „OK“ setzen möchten, können Sie eine der folgenden Schreibweisen verwenden:

```
pushbutton1.caption=”OK”
```

oder

```
me.caption=”OK”
```

Der Vorteil bei der Verwendung von *Me* ist, dass Ihr Code universeller wird. Sie können ihn bei jedem anderen PushButton-Steuerelement ohne Änderung einsetzen.

Der Befehl *Self* verweist auf das Elternfenster des Steuerelements. Sie können ihn in einem Event-Handler für jedes Steuerelement eines Fensters verwenden. Folgende Zeilen sind äquivalent:

```
FindDialog.title”Find”
Self.Title=”Find”
Title=”Find”
```

Im ersten Beispiel wird die Name-Eigenschaft des Fensters verwendet; im zweiten Beispiel erreicht der *Self*-Befehl das gleiche, ohne den Namen des Fensters explizit verwenden zu müssen. Das letzte Beispiel nutzt die Tatsache aus, dass das Elternfenster verwendet wird, wenn Sie den Code in einem Steuerelement innerhalb des Fensters verwenden.

Werte aus Eigenschaften auslesen

Sie können den Wert einer Eigenschaft fast genauso auslesen, wie Sie Werte zuweisen. Dazu muss nur auf der linke Seite des Gleichheitszeichens bestimmt werden, wohin der Wert übernommen werden soll, während auf der rechten Seite die Eigenschaft des Objekts steht, deren Wert ausgelesen werden soll. Soll beispielsweise einer Variablen X die Caption-Eigenschaft des pushbutton1 zugewiesen werden, so sieht die Syntax dazu so aus:

```
x=pushbutton1.caption
```

Ebenso funktioniert es, wenn Sie den Wert einer Eigenschaft eines Objektes in einem anderen Fenster auslesen wollen (analog zum oben geschilderten Fall):

```
x=window2.pushbutton1.caption
```

Eine Eigenschaft des Fensters, in dem sich das Objekt befindet, dessen Code ausgeführt wird, kann so ausgelesen werden:

```
x=title
```

Auch hier muss der Name des Fensters dann nicht explizit angegeben werden.

Werte in Variablen

Wenn Werte gespeichert werden sollen, die nichts mit Eigenschaften von Objekten zu tun haben, dann verwenden Sie Variablen. Eine Variable ist ein Bereich im Speicher, dem Sie einen Namen geben. Den Namen wählen Sie so, dass er eine Aussage über die Funktion der Variable liefert. Wenn Sie das Alter einer Person in Tagen seit ihrer Geburt berechnen wollen, würden Sie eine Variable verwenden, die Sie „Tage“ nennen. Variablen können Namen beliebiger Länge haben. Die Namen müssen aber mit einem Buchstaben anfangen, wobei auch Umlaute verwendet werden dürfen. Ansonsten sind folgende Zeichen gestattet: A-Z, a-z, 0-9 und der Unterstrich („_“). Variablennamen sind nicht case-sensitiv. Das bedeutet, dass zwischen Großbuchstaben und Kleinbuchstaben kein Unterschied besteht. x und X ist für REALbasic dieselbe Variable.

Sie können Werte in Variablen speichern und aus Variablen auslesen, so wie das auch bei Eigenschaften möglich ist. Um einen Wert auszulesen, muss die Variable auf der rechten Seite des Gleichheitszeichens (=) stehen. Soll die Beschriftung eines Knopfes auf den Inhalt einer Variablen namens „buttonTitle“ gesetzt werden, dann schreibt man das so:

```
PushButton1.Caption=buttonTitle
```

Als erstes legen Sie die Variable an und weisen ihr einen Datentyp zu. Wie auch die eingebauten Objekt-Eigenschaften haben Variablen einen Datentyp. Bevor Sie eine Variable einsetzen, muss ihr Datentyp mit dem `Dim`-Ausdruck bekannt gegeben werden. `Dim` ist die Abkürzung für Dimension. Der Ausdruck zielt darauf ab, Platz für die Variable zu schaffen. Alle Ihre `Dim`-Ausdrücke müssen innerhalb einer Methode vor den ersten Code-Zeilen erscheinen. Sobald der Datentyp einer Variable einmal deklariert ist, verbleibt die Variable für die Laufzeit der Methode im Speicher.

In diesem Beispiel würden Sie folgenden Ausdruck verwenden:

```
Dim buttonTitle as String
```

Damit wird die Variable `buttonTitle` im Speicher angelegt und REALbasic wird vermittelt, dass die Variable nur String-Werte speichern kann. Es kann allerdings nur ein String gespeichert werden, nicht mehrere. Nachdem Sie die Variable erstellt haben, geben Sie ihr einen Wert und weisen Ihren Wert einer anderen Variable oder Eigenschaft zu. So können Sie zum Beispiel schreiben:

```
buttonTitle="Save"
```

Nachdem Sie der Variable einen Wert zugewiesen haben, können Sie diesen der Eigenschaft eines bestehenden Objekts zuordnen. Das heißt, dass Sie die Variable einer Eigenschaft vom Typ String zuordnen können.

Dies wird im folgenden Beispiel erreicht:

```
Pushbutton1.Caption=buttonTitle
```

Soll umgekehrt der Wert, der in der Caption-Eigenschaft des Knopfes steht, in einer Variablen gespeichert werden, dann schreibt man das so:

```
buttonTitle=pushButton1.Caption
```

Im folgenden Beispiel wird i als Variable vom Typ Integer deklariert:

```
Dim i as integer
```

Wenn mehrere Variablen vom gleichen Typ deklariert werden sollen, dann kann man das in einem Aufwasch tun:

```
Dim i,j,k as integer
Dim Name,Adresse as String, Schuhgroesse as Single
```

Wenn Sie eine Variable mit der Dim-Answeisung deklarieren, können Sie dieser auch einen Wert zuweisen. Dazu geben Sie nach dem Datentyp ein Gleichheitszeichen und einen Wert an. Die folgenden Ausdrücke sind gültig:

```
Dim i as Integer = 3
Dim Name as String = "Igor", Adresse as String = "Baumallee 15"
```

Sie können dies auch mischen, wie folgender Ausdruck zeigt:

```
Dim a as Integer, b as Integer = 15
```

Mit diesem Ausdruck wird die Variable a als Integer deklariert, erhält jedoch keinen Wert, während b den Wert 15 zugewiesen bekommt.

Auch der folgende Ausdruck ist gültig:

```
Dim a,b,c as Integer = 15
```

Mit diesem Ausdruck werden a,b und c als Integer deklariert und bekommen den Wert 15 zugewiesen. Auch wenn dies eine gültige Zuweisung ist, sollten Sie mit Zuweisungen für mehrere Variablen in einem Dim-Ausdruck vorsichtig umgehen, da man schnell den Überblick verliert.

So wie bei Eigenschaften, kann man nur kompatible Datentypen einander zuweisen. Folgendes Beispiel führt zu einem Fehler:

```
Dim x, z as integer
Dim y as string
x=1
y="Hello"
z=x+y
```

x ist eine Zahl, y aber eine Zeichenkette. Da verschiedene Datentypen nicht addiert werden können, kommt es zu einer Fehlermeldung.

Objekte deklarieren

Sie kennen bereits die Datentypen Integer, String, Boolean, Single, Double und Color. Variablen können aber auch mit bestimmten Objekttypen deklariert werden. Zum Beispiel gibt es ein REALbasic-Objekt namens Date, das Sie verwenden, wenn Sie ein Datum oder eine Uhrzeit speichern wollen.

Technisch gesehen ist Date eine REALbasic-Klasse. Den Klassen ist ein eigenes Kapitel gewidmet, in dem Sie mehr darüber erfahren.

Wenn Sie eine Date-Variable verwenden wollen, wird die eingebaute Klasse als Vorlage verwendet. Eine solche Vorlage bietet genug Platz, um die Eigenschaften des Objekts und in manchen Fällen auch die Methoden zu speichern, die zu einem Objekt der Klasse gehören.

Sie erzeugen eine Instanz dieser Vorlage, die Sie dann verwenden können. Die Instanz ist die Variable, die Sie in Ihrem Code benutzen. Die Klasse selbst können Sie nicht benutzen, da diese als Vorlagen für Instanzen dient. Objekte einer Klasse werden instanziiert, weshalb man bei einem Objekt auch immer eine Instanz einer Klasse meint.

Die Date-Klasse, zum Beispiel, besitzt Eigenschaften für das Jahr, den Monat, den Tag, Minuten und Sekunden. Zusätzlich besitzt sie Eigenschaften, die das Datum formatieren, wie ShortDate (kurzes Datum), AbbreviatedDate (abgekürztes Datum) oder LongDate (langes Datum).

Eine neue Variable dieses Typs erzeugen Sie wie jede andere Variable auch.

```
Dim Geburtsdatum as Date
```

Dieser Ausdruck hat allerdings noch keine Instanz der Date-Klasse erzeugt. Der folgende Ausdruck würde also nicht funktionieren:

```
Dim Geburtsdatum as Date
Geburtsdatum.Year = 1996
```

Bevor Sie auf Eigenschaften der Date-Variablen zugreifen können, müssen Sie mit Hilfe des New-Operators eine Instanz erzeugen. Sprich, ein Objekt der Klasse Date instanziieren:

```
Dim Geburtstag as Date
Geburtstag = New Date
```

Nun können Sie auf die Eigenschaften der Date-Variable zugreifen, zum Beispiel ein anderes Datum setzen:

```
Dim Geburtstag as Date
Geburtstag=New Date
Geburtstag.Year=1996
Geburtstag.Month=6
Geburtstag.Day=23
```

Wenn Sie nun die ShortDate Eigenschaft auslesen, hätte diese den Wert "6/23/96". Es gibt eine noch eine kürzere Syntax, um die Deklaration und Instanziierung zusammenzufassen:

```
Dim Geburtstag as New Date
Geburtstag.Year = 1996
Geburtstag.Month = 6
Geburtstag.Day = 23
```

Wenn Sie die Instanzierung auslassen, wird der REALbasic-Compiler eine Fehlermeldung ausgeben, wenn Sie auf eine Eigenschaft der Variable zugreifen oder eine Methode aufrufen.

Verwenden von Arrays

Mit der Dim-Anweisung können auch Arrays angelegt werden. Ein Array ist eine Variable, in der man mehrere Werte des gleichen Typs speichern kann, die als Elemente bezeichnet werden.

Ein Array wird angelegt, indem man in Klammern den Index des letzten Elements im Array angibt. Die Anzahl der Werte, die man im Array ablegen kann, ist tatsächlich allerdings um eins größer als die Anzahl, die man bei der Dim-Anweisung angibt. REALbasic legt nämlich nullbasierte Arrays an, die immer ein Element mit der Nummer 0 enthalten.

Die Anweisung:

```
Dim aNames (10) as string
```

legt ein String-Array mit elf Elementen an.

Die Anweisung:

```
Dim aNames(0) as string
```

legt einen String-Array mit einem Element an, Element Null. Sie können auf dieses Element genau so wie auf jedes andere Element mit einem positiven Index lesend und schreibend zugreifen.

Wenn Ihnen die Größe des Arrays zum Zeitpunkt der Deklaration nicht bekannt ist, können Sie ein sogenanntes Null-Array deklarieren. Ein Null-Array ist ein Array, das keine Elemente enthält. Dazu geben Sie bei der Deklaration einen Index von -1 an oder lassen den Index weg.

Die Anweisungen

```
Dim Vornamen(-1) as String
```

oder

```
Dim Vornamen() as String
```

erzeugen ein Array, das keine Elemente enthält. Sie können die Größe zu einem späteren Zeitpunkt mit Hilfe der ReDim-Anweisung ändern oder das Array mit der Append-Methode „wachsen“ lassen.

Sie können auch mehrdimensionale Arrays mit REALbasic anlegen. Die maximale Anzahl an Dimensionen in einem multidimensionalen Array beträgt 20. Um ein mehrdimensionales Array anzulegen, müssen Sie für jede Dimension eine Indexnummer angeben.

Die Anweisung

```
Dim aNames (2,10) as string
```

legt zum Beispiel ein Array mit drei Zeilen und elf Spalten an.

Sie können in Dim-Befehlen globale Konstanten verwenden, um die Größe eines Arrays festzulegen. Folgende Deklaration verweist auf die globale Konstante „nLanguages“, die in einem Modul definiert wurde:

```
Dim aControl(10,nLanguages) as String
```

nLanguages ist die Anzahl der vom Programm unterstützten Sprachen und wird an vielen Stellen im Programm verwendet. Wenn sich etwas ändert, müssen Sie nur ihre Definition im Modul ändern. Weitere Informationen finden Sie im Abschnitt „Einfügen von Konstanten in ein Modul“ auf Seite 232.

Ein einzelnes Element eines Arrays wird mit der Nummer angesprochen, die seinen Platz im Array definiert. Hier wird in der ersten Spalte der ersten Zeile des Arrays der Name „Frank“ abgelegt:

```
aNames(0,0)="Frank"
```

Die Ubound-Funktion liefert den Index des letzten Elements in einem eindimensionalen Array. Die Anzahl der Elemente ist um eins größer als die ausgegebene Zahl, da das Array ein „nulltes“ Element besitzt. Bei einem mehrdimensionalen Array liefert Ubound den Index des letzten Elements der Dimension, die Sie dafür festlegen. Wenn Sie keine Dimension angeben, wird der entsprechende Wert der ersten Dimension ausgegeben. Die erste Dimension trägt die Nummer 1. Das folgende Beispiel ergibt für die Variable i den Wert 5 und für die Variable j den Wert 3.

```
Dim i,j as integer
Dim aNames(5,3) as string
i=Ubound(aNames)
j=Ubound(aNames,2)
```

Arrays initialisieren

Nachdem Sie ein Array deklariert haben, können Sie ihm initiale Werte zuweisen. Dies können Sie mit der Array-Funktion oder auch mit der individuellen Zuweisung der Elemente, wie im Beispiel gezeigt:

```
Dim Namen(2) as String
//separate Zuweisungen
Namen(0)="Fred"
Namen(1)="Ginger"
Namen(2)="Stanley"
```

Im folgenden Beispiel wird die Array-Funktion verwendet, um genau dasselbe zu erreichen. In diesem Beispiel muss die Größe des Arrays nicht angeben werden. Die Array-Funktion fügt die Elemente selbständig dem Array hinzu:

```
Dim Namen(-1) as String
Namen=Array("Fred","Ginger","Stanley")
```

Wenn Sie das Array mit einer fixen Größe deklarieren, aber nicht genügend Parameter angeben, um alle Elemente zu füllen, beginnt die Array-Funktion mit dem Element Null und füllt die folgenden Elemente mit den angegebenen Werten.

Array-Zuweisungen

Zwei Arrays mit kompatiblen Datentypen können einander zugewiesen werden. Hier ist ein Beispiel:

```
Dim NamenA(2),NamenB(2) as String
NamenA=Array("Fred","Ginger","Stanley")
NamenB=NamenA
```

Dieser Ausdruck weist den ersten drei Elementen in NamenB die Werte von NamenA zu. Würde NamenB weniger Elemente besitzen als NamenA, würden zusätzliche Elemente hinzugefügt werden, damit alle Elemente von NamenA einem Element in NamenB zugewiesen werden können. Ein Beispiel:

```
Dim NamenA(3),NamenB(2) as String
NamenA(0)="Fred"
NamenA(1)="Ginger"
NamenA(2)="Tommy"
NamenA(3)="Woody"
NamenB=NamenA
```

Nach der Ausführung dieses Codes hat NamenB vier Elemente statt vorher drei, um auch den Wert „Woody“ aufnehmen zu können.

Die Größe von Arrays anpassen

Mehrere Befehle sind für die Anpassung der Größe von Arrays verantwortlich:

- **Append:** Append fügt dem Array ein Element hinzu und vergrößert die Anzahl der Elemente im Array um die Zahl 1. Wenn Sie Append aufrufen, übergeben Sie den Wert, den Sie dem Array hinzufügen wollen. Die Anweisung:
  ```
  aNames.Append "Dave"
  ```
  fügt dem Array aNames ein Element hinzu und setzt den Wert für dieses Element auf den String „Dave“.

  Append lässt sich nur auf eindimensionale Arrays anwenden.
- **Insert:** Die Insert-Methode fügt dem Array ein neues Element an der von Ihnen definierten Stelle hinzu. Insert benötigt dafür zwei Parameter: den Index des Elements, das eingefügt werden soll und den Wert des neuen Elements. Zum Beispiel:
  ```
  aNames.Insert 9,"Hal"
  ```
  Nach der Ausführung diese Befehls wäre „Hal“ der Wert für aNames(9). Das alte Element (9) wäre um einen Index nach oben an die Stelle von Element (10) gewandert usw. Genauso wie durch den Befehl Append vergrößert sich die Anzahl der Elemente im Array um die Zahl 1.

  Insert lässt sich ebenfalls nur auf eindimensionale Arrays anwenden.
- **Remove:** Die Remove-Methode löscht das Element mit dem angegebenen Index. Die Anweisung:
  ```
  aNames.Remove 9
  ```
  löscht das Element mit Index 9, verringert die Anzahl der Elemente im Array um die Zahl 1. Alle Elemente, die dem gelöschten Element folgen, rücken um eine Stelle nach unten. Diese Methode lässt sich ebenfalls nur auf eindimensionale Arrays anwenden.
- **Redim:** Die Redim-Methode verändert die Größe eines bestehenden Arrays. Übergeben Sie Redim die neue Anzahl der Array-Elemente. Der Datentyp wurde schon durch die Dim-Anweisung bestimmt. Die Anweisung:
  ```
  Redim aNames(100)
  ```
  vergrößert das Array auf 101 Elemente. Redim lässt sich sowohl auf eindimensionale als auch auf mehrdimensionale Arrays anwenden. Bei mehrdimensionalen Arrays können Sie nur bereits bestehende Dimensionen in ihrer Größe variieren. Dimensionen können nicht hinzugefügt oder gelöscht werden.

  Ein Unterschied zwischen Dim und Redim besteht darin, dass Dim nur Integer oder globale Konstanten als Werte akzeptiert, während Redim alle Ausdrücke toleriert, die einen Integer-Wert ergeben. Das kann zum Beispiel eine

benutzerdefinierte Funktion sein, die einen Integer-Wert ergibt. Damit können Sie die Größe Ihres Arrays im Verlaufe einer Anwendung dynamisch gestalten.

Konvertieren zwischen Array und Variable

Es gibt zwei Funktionen, um ein Array in eine Variablen zu konvertieren oder aus dem Inhalt eines Strings ein Array zu erzeugen.

- **Split:** Der Split-Funktion wird ein String übergeben, der in ein eindimensionales Array konvertiert wird. Dies geschieht, indem der String an einem anzugebenen Trennzeichen zerteilt wird. Das voreingestellte Trennzeichen ist ein Leerzeichen. Hier ein Beispiel, bei dem das Komma als Trennzeichen verwendet wird:

```
Dim aNames(-1) as String //Die Split-Funktion ändert die Größe
Dim s as String
s="Juliet,Taylor,Casting"
aNames=Split(s,",")
Nach dem Aufruf von Split besitzt aNames drei Elemente.
aNames(0)="Juliet"
aNames(1)="Taylor"
aNames(2)="Casting"
```

- **Join:** Die Join-Funktion erzeugt aus einem eindimensionalen Array einen String. Dabei kann ein Trennzeichen angegeben werden, das im String zwischen die einzelnen Array-Elemente gesetzt werden soll. Voreingestellt ist das Leerzeichen. Im folgenden Beispiel geht es um ein Array, das Vor- und Nachnamen und die Telefonnummer einer Person enthält. Mit Hilfe der Join-Funktion können Sie daraus einen String erzeugen, der all diese Informationen enthält.

```
Dim aNames(2) as String
Dim s as String
aNames(0)="Anthony"
aNames(1)="Aardvark"
aNames(2)="737-8946"
s=Join(aNames,",")
```

  Dieser Programmcode liefert in der Variablen s den String `"Anthony,Aardvark,737-8946"`.

  Würde der Aufruf folgendermaßen lauten:

```
s=Join(aNames)
```

  Dann wäre das Ergebnis ein String mit dem Inhalt `"Anthony Aardvark 737-8946"`.

Dictionaries

Unter einem Dictionary (Wörterbuch) versteht man ein REALbasic-Objekt, das aus einer Liste von „Schlüssel – Wert"-Paaren besteht. Das besondere an Dictionaries ist, dass sowohl der „Schlüssel" als auch der dazugehörige Wert dem Datentyp Variant angehören. Dies bedeutet, dass ein Dictionary Werte von verschiedenen Datentypen enthalten kann und der „Schlüssel" nicht zwingend ein Integer-Wert sein muss. Sie können in einem Dictionary einen Wert abfragen, indem Sie entweder über den „Schlüssel" darauf zugreifen oder die Position des Wertes in der Reihenfolge der Einträge angeben. Der Sprachreferenzeintrag zum Thema Dictionaries zeigt ein interessantes Beispiel, in dem Farben die Funktion des „Schlüssels" übernehmen.

Mathematische Operatoren

Mathematische Berechnungen kommen bei der Programmierung häufig vor. REALbasic kennt die üblichen mathematischen Operationen:

Operation	Operator	Beispiel
Addition	+	2 + 3 = 5
Subtraktion	-	3 - 2 = 1
Multiplikation	*	3 * 2 = 6
Fließkommadivision	/	6 / 4 = 1.5
Integer-Division	\	6 \ 4 = 1
Modulo	Mod	6 Mod 3 = 0 6 Mod 4 = 2

Es gibt außerdem noch eine Menge mathematischer Funktionen, die Sie in der Sprachreferenz finden.

REALbasic verwendet die übliche mathematische Ausführungsfolge. Geklammerte Elemente werden zuerst berechnet. REALbasic beginnt mit der Bearbeitung der Elemente, die innerhalb der größten Anzahl von Klammern stehen (also sozusagen von innen nach außen). Danach erfolgen Multiplikationen oder Divisionen von links nach rechts und am Schluss Additionen und Subtraktionen. Die folgenden Gleichungen liefern alle unterschiedliche Werte, da die Klammern unterschiedlich gesetzt wurden:

Gleichung	Ergebnis
2+3*(5+3)	26
(2+3)*5+3	28
2+(3*5)+3	20

Konstanten

Eine Konstante ist eine Variable, die einen festen Wert enthält. Dieser Wert kann nicht nachträglich verändert werden, sondern wird beim Erzeugen der Konstante zugewiesen. Das Erzeugen von Array-Konstanten ist nicht möglich.

In REALbasic können Sie Konstanten für Fenster, Module und Objekte erzeugen, die Sie Ihrem Projekt hinzugefügt haben (Fenster, Module und Klassen werden später in diesem Handbuch behandelt). Sie können auch lokale Konstanten in einer Methode erzeugen.

Jede Konstante hat einen Gültigkeitsbereich (Scope). Dieser Gültigkeitsbereich legt fest, welcher Teil des Programms auf die Konstante zugreifen kann. Wenn Sie eine Konstante in einer Methode erzeugen, kann nur innerhalb dieser Methode auf sie zugegriffen werden. Der Gültigkeitsbereich beschränkt sich also auf diese Methode, andere Teile der Applikation kennen die Konstante nicht. Wenn eine andere Methode, eine Konstante mit gleichem Namen erzeugt, so ist dies eine völlig andere Konstante. Die Konstanten stehen in keiner Beziehung, da ihr Gültigkeitsbereich sich auf die jeweilige Methode beschränkt.

Um eine lokale Konstante zu erzeugen, verwenden Sie die Const-Answeisung und eine Zuweisung.

```
Const <EineKonstante> = <Wert>
```

Sie brauchen für eine Konstante keinen Typ anzugeben, wie Sie es bei einer Dim-Anweisung tun müssten. Der folgende Ausdruck ist gültig:

```
Const BackGroundColor=&cff0000
```

REALbasic erkennt, dass es sich um eine Konstante des Typs Color handelt. Sie können nun die Konstante BackgroundColor jeder Variable oder Eigenschaft zuweisen, die ebenfalls vom Typ Color ist. Ein Beispiel:

```
Const Accept="OK"
bevelbutton1.caption=Accept
```

Wenn die Applikation gestartet wird, ist die Beschriftung des Buttons "OK".

Konstanten eines Fensters, eines Moduls oder einer Klasse haben mehrere mögliche Gültigkeitsbereiche.

- **Global** (nur für Modulkonstanten sinnvoll): Auf eine globale Konstante kann innerhalb des gesamten Projekts zugegriffen werden. Globale Konstanten können die Pflege eines Projekts stark vereinfachen, wenn alle an einer zentralen Stelle deklariert werden. Wenn Sie dann eine Konstante ändern müssen, wissen Sie sofort, wo sie deklariert ist und dass sich die Änderung auf das gesamte Projekt auswirken wird.

  Globale Konstanten sind ein praktisches Werkzeug, wenn Sie mehrere Sprachversionen Ihres Projekts erzeugen wollen. Mehr dazu finden Sie im Abschnitt „Lokalisieren mit Konstanten" auf Seite 234.
- **Öffentlich:** Die Konstante kann innerhalb des gesamten Projekts verwendet werden. Um die Konstante innerhalb des Objekts zu verwenden, das die Konstante „besitzt", greifen Sie über die Bezeichnung auf diese zu, wie bei einer globalen Konstante.

  Um eine öffentliche Konstante aus einem anderen Objekt heraus zu verwenden, müssen Sie die Punkt-Notation verwenden, also „Objektname.Konstantenname". Nehmen wir an, es gibt ein Fenster mit dem Namen „SaveChangesWindow", welches eine öffentliche Konstante „Accept" besitzt. In den Methoden und im EventHandler des Fensters würden Sie die Konstante einfach über deren Bezeichnung verwenden. Wenn Sie sie aber in einer Methode, die nicht zu dem Fenster gehört, verwenden wollen, müssen Sie "SaveChangesWindow.Accept" schreiben.
- **Geschützt:** Geschützte Konstanten können nur in den Objekten verwendet werden, für die sie erzeugt wurden. Wenn Sie beispielsweise eine geschützte Konstante für ein Fenster erzeugen, können Sie diese nur innerhalb der Methoden und EventHandler dieses Fensters und seiner Steuerelemente verwenden. Der Zugriff erfolgt über den Namen des Konstanten.
- **Privat:** Eine private Konstante ist einer geschützten Konstante sehr ähnlich. Allerdings kann diese nur in Code verwendet werden, der direkt zu dem Objekt gehört. Sie kann nicht in Objekten verwendet werden, die von dem Objekt abgeleitet worden sind. Der Zugriff erfolgt über den Namen der Konstanten.

Konstanten können nur im Fenster des Code-Editors erzeugt werden. Weiterhin können Konstanten nur für Fenster, Module und Klassen erzeugt werden. Konstanten werden nicht im Projektfenster abgelegt.

Wenn Sie beispielsweise für das Standardfenster „Fenster1" eine Konstante erzeugen möchten, öffnen Sie den Code-Editor und führen folgende Schritte aus:

- Wählen Sie den Menüpunkt **Neu/Neue Konstante**. Daraufhin erscheint ein Dialog.
- Bestimmen Sie den Namen, den Typ und den Wert.
- Wählen Sie einen Gültigkeitsbereich aus.

  Für Fenster haben Sie die Wahl zwischen Öffentlich, Geschützt und Privat.

  Die Liste unter dem PopupMenu enthält Konstanten für die Lokalisierung Ihrer Applikation. Sie können für die verschiedenen Sprachen unterschiedliche Werte festlegen, sogar für unterschiedliche Plattformen. Lesen Sie dazu auch den Abschnitt „Eine Konstante einem Modul hinzufügen" auf Seite 232.
- Klicken Sie auf „OK", um die Konstante zu speichern.

  Jetzt können Sie die Konstante unter Berücksichtigung ihres Gültigkeitsbereichs in Ihrem Projekt verwenden. Sie können sogar im Eigenschaftenfenster diese Konstante einer Eigenschaft zuweisen, indem Sie als Wert für die Eigenschaft ein Nummernzeichen („#") gefolgt vom Namen der Konstanten verwenden. Falls Sie dieser Konstanten in Abhängigkeit von Sprache und Plattform verschiedene Werte zugewiesen haben, werden beim Erzeugen des ausführbaren Programms automatisch die zu den Compiler-Einstellungen passenden Werte eingesetzt.

Reservierte Wörter

Die folgenden Schlüsselwörter können nicht als Variablennamen verwendet werden, da sie schon Teil REALbasic-Sprache sind:

Tabelle 4. Reservierte Wörter

And	ElseIf	Loop	Select
Array	End	Me	Self
As	Event	Mod	Shared
Boolean	Exception	Module	Single
ByRef	Exit	Namespace	Static
ByVal	False	New	Step
Call	Finally	Next	String
Case	For	Nil	Sub
Catch	Function	Not	Then
Class	GoTo	Object	To
Color	Handles	Of	True
Const	If	Or	Try
DebugPrint	Implements	Private	Until
Declare	In	Protected	Wend
Dim	Inherits	Public	While
Do	Inline68K	Raise	#bad
Double	Integer	Redim	#else
Downto	Interface	Rem	#endif
Each	Isa	Return	#if
Else	Lib	Return	#pragma

Ausführen von Instruktionen in Methoden

Eine Methode besteht aus einer oder mehreren Anweisungen, die ausgeführt werden, um ein bestimmtes Resultat zu erreichen. REALbasic kennt viele eingebaute Methoden. Beispielsweise führt der Aufruf der Methode „Quit“ dazu, dass Ihr Programm sich beendet. Einige Objekttypen (Klassen) haben eingebaute Methoden. Beispielsweise hat die Klasse ListBox eine Methode namens AddRow, die zu einer ListBox eine neue Zeile hinzufügt (wie der Name schon andeutet). Sie können aber auch selbst eigene Methoden definieren. Eigenen Methoden gibt man einen Namen, der ihre Funktion beschreibt. Auch hier kann man einen Namen beliebiger Länge verwenden, der mit einem Buchstaben beginnen muss und ansonsten Zeichen von a-z, A-Z, 0-9, Umlaute und den Unterstrich („_“) enthalten darf.

Im folgenden sehen Sie eine einfache Methode, die 1999 das Alter einer Person in Tagen berechnet, die 1960 geboren ist (nicht exakt, da Schaltjahre ignoriert werden):

```
Dim yearBorn, thisYear, daysOld as Integer
yearBorn=1960
thisYear=1999
daysOld=(thisYear-yearBorn)*365
```

Methoden können natürlich erheblich komplexer als in diesem Beispiel sein. Es gibt drei verschiedene Stellen, an denen Sie ihren Programmcode unterbringen können. Dazu mehr im nächsten Kapitel.

Kommentare und Dokumentation

Es ist sehr wichtig, dass Sie Ihre Quelltexte gut dokumentieren, damit Sie auch nach längerer Zeit noch mühelos nachvollziehen können, was Sie da programmiert haben. Sie sollten außerdem die Steuerelemente und andere Objekte

logisch und konsequent benennen. Wenn jemand anders Änderungen an Ihrem Quelltext vornehmen muss, ist eine gute Dokumentation unverzichtbar.

Kommentare können in eigenen Zeilen stehen oder rechts an eine Code-Zeile anschließen. Die Kommentare sind nur Bestandteil des Quelltextes (also des Projekts), nicht der fertig gebauten Anwendung.

Kommentare werden entweder mit einem einfachen Anführungszeichen (‘), mit zwei Schrägstrichen (//) oder mit dem Wort REM (für remark=Bemerkung) begonnen:

```
//Anlegen der benötigten Variablen
Dim yearBorn, thisYear, daysOld as Integer
yearBorn=1960 //Geburtsjahr merken
thisYear=1999 //aktuelles Jahr merken
//Das Alter in Tagen ausrechnen
daysOld=(thisYear-yearBorn)*365
```

Kommentare werden gemäß Vorgabe angezeigt. Sie können über die REALbasic-Voreinstellungen eine andere Farbe für die Kommentare auswählen.

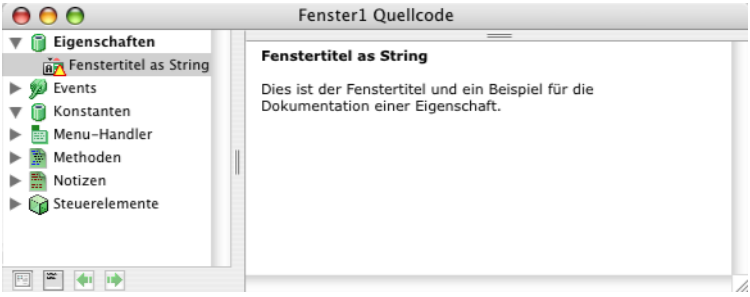
Falls Sie mehrere aufeinanderfolgende Zeilen auskommentieren wollen (um den betreffenden Programmcode auszuklammern), selektieren Sie mit der Maus die entsprechenden Zeilen und drücken dann ⌘-„“ (Mac) oder Strg-„“ (Windows). Mit derselben Tastenkombination können Sie die Kommentare wieder in ausführbaren Code zurückverwandeln. Diese Technik ist vor allem dann nützlich, wenn Sie vorübergehend mehrere Quelltextzeilen zu Testzwecken auskommentieren möchten.

Eigenschaften dokumentieren

Wenn Sie eine Eigenschaft für ein Fenster, ein Modul oder eine Klasse erzeugen, können Sie diese im Code-Editor dokumentieren. REALbasic zeigt automatisch die Deklaration in Fettschrift an und bietet Ihnen die Möglichkeit, darunter eigenen Text einzufügen.

Der Text, den Sie an dieser Stelle eingeben, ist nicht ausführbar. Selbst wenn es gültiger REALbasic Code ist.

Weitere Informationen finden Sie im Abschnitt „Weitere Eigenschaften für ein Fenster anlegen“ auf Seite 205.

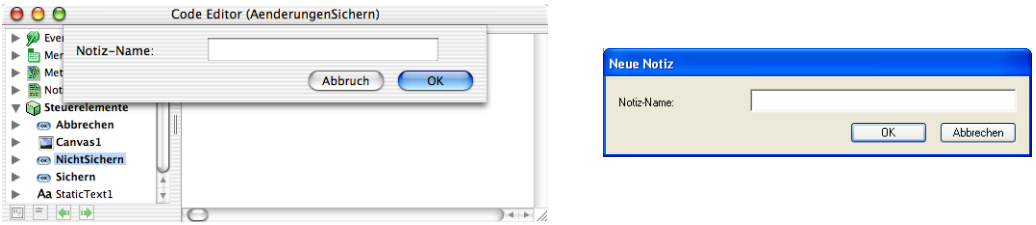


Notizen eingeben

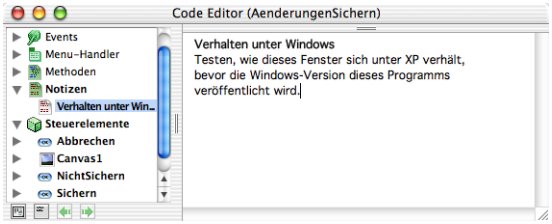
Sie können Ihren Code (oder Ihre Anwendung) auch zentral mit Hilfe der Notiz-Option dokumentieren. Jeder Code-Editor besitzt einen eigenes Notiz-Element, in dem Sie Notizen aufbewahren können. Genau wie Kommentare werden auch Notizen nicht compiliert und werden nicht in die fertige Anwendung übernommen. Auch wenn die Notizen im Code-Editor auftauchen, sollten Sie nicht auf die Idee kommen, in Ihrem Code auf diese zu verweisen.

Eine Notiz fügen Sie genauso wie ein Objekt ein. Rufen Sie **Bearbeiten/Neue Notiz** auf. Es erscheint eine Dialogbox (unter Mac OS X ein Sheet-Fenster), in der Sie die Notiz benennen können.

Abb. 137: Das Notiz-Sheet-Fenster unter Mac OS X und die Dialogbox unter Windows



Jetzt erscheint die Notiz im Browser unter „Notizen“ und kann im Editor bearbeitet werden.



Werte an Methoden übergeben

Einige der in REALbasic eingebauten Methoden benötigen zusätzliche Informationen, um ihre Aufgaben erfüllen zu können. Diese Informationen nennt man Parameter. Parameter werden an Methoden übergeben, indem man sie rechts neben den Namen der Methode schreibt. Im folgenden Beispiel wird mit der Methode AddRow an die ListBox1 eine Zeile angehängt. Dazu muss man noch einen Wert übergeben, nämlich den Inhalt der Zeile. In diesem Fall ist es der Text „Januar“:

```
ListBox1.AddRow "Januar"
```

Sie können den Parameter auch in Klammern setzen. Der folgende Ausdruck ist äquivalent:

```
ListBox1.AddRow ("Januar")
```

Benötigt eine Methode mehr als einen Parameter, trennt man die Parameter mit Kommata voneinander. In der Klasse ListBox gibt es eine Methode InsertRow. Diese erlaubt es, neue Zeilen in eine bestehende ListBox einzufügen. Dazu muss man InsertRow mitteilen, an welcher Stelle die neue Zeile eingefügt werden soll und welcher Wert dann in dieser Zeile stehen soll:

```
ListBox1.InsertRow 3, "Januar"
```

Auch mehrere Parameter können in Klammern gesetzt werden:

```
ListBox1.InsertRow (3, "Januar")
```

Auch Variablen können als Parameter verwendet werden. Dabei wird der momentane Wert der Variablen an die Methode übergeben:

```
Dim Monat as String
Monat="Januar"
ListBox1.InsertRow 3, Monat
```

Arrays als Parameter übergeben

Ein eindimensionales Array kann als Parameter beim Aufruf einer Unterroutine oder Funktion übergeben werden. Um festzulegen, dass es sich bei einem Parameter um ein Array handeln soll, versehen Sie ihn mit leeren Klammern:

```
Names () as String
```

Obige Deklaration verwenden Sie, wenn Sie einer Unterroutine ein Array aus Strings übergeben möchten. Da Sie die Anzahl der Elemente nicht angeben müssen, können Sie an verschiedenen Stellen im Code eine unterschiedliche Anzahl von Elementen übergeben.

Wenn Sie im Quelltext das Array als Parameter übergeben, lassen Sie die Klammern weg, z.B.:

```
PrintLabels (Names)
```

PrintLabels ist der Name der Methode, die das String-Array als Parameter erwartet.

Rückgabe von Werten aus Methoden

Manche Methoden liefern Ergebnisse zurück. Das bedeutet, dass ein Wert von der Methode in der Zeile, in der sie aufgerufen wurde, zurückgegeben wird. Beispielsweise liefert die in REALbasic eingebaute Methode **Ticks** die Anzahl von Ticks (60stel Sekunden), die seit dem Einschalten des Rechners vergangen sind. Diesen Rückgabewert können Sie ganz normal z.B. einer Variablen zuweisen. Im Beispiel werden die Ticks der Variablen x zugewiesen:

```
x=Ticks
```

Einige Methoden benötigen Parameter und geben einen Wert zurück. Beispielsweise wird die Funktion Chr verwendet, um das ASCII-Zeichen mit einer bestimmten Nummer zurückzugeben. Wird eine Methode aufgerufen, die einen Wert zurückliefert, müssen die Parameter in Klammern angegeben werden. Wird der Chr-Funktion eine 13 übergeben – der ASCII-Wert für Wagenrücklauf (Carriage-Return) – dann liefert sie das ASCII-Zeichen für Wagenrücklauf zurück:

```
x=Chr(13)
```

Die Klammern sind erforderlich, weil der Rückgabewert auch direkt als Parameter für eine andere Methode eingesetzt werden könnte. Im folgenden Beispiel kann man das sehen:

```
ListBox1.InsertRow 3, Str(Len("Hello"))
```

In der ListBox1 wird eine Zeile an der dritten Position eingefügt. In dieser Zeile soll die Länge der Zeichenkette “Hello”, also die Zahl 5 erscheinen. Die ListBox benötigt dazu eine Zeichenkette, die mit der Str-Funktion aus dem numerischen Ergebnis der Len-Funktion (die die Länge einer Zeichenkette liefert) erzeugt wird.

Methoden, die ein Ergebnis zurückliefern, bezeichnet man als Funktionen. In der Sprachreferenz sind solche Methoden als Funktionen markiert.

Parameter als Wert oder als Referenz übergeben

Standardmäßig werden Parameter als Werte an Methoden übergeben. Die Methode erhält eine Kopie der Daten und kann Operationen mit diesen Daten ausführen.

Parameter, die als Werte übergeben werden, werden innerhalb einer Methode als lokale Variablen behandelt – genauso wie Variablen, die mit dem Ausdruck `Dim` angelegt werden. Dies bedeutet, dass Sie die Werte der Parameter selbst verändern können, ohne vorher den Parameter einer lokalen Variable zuzuweisen. Wenn Sie zum Beispiel mit dem Parameter “x” einen Wert übergeben, dann können Sie den Wert von x inkrementieren oder dekrementieren anstatt ihn einer lokalen Variable zuzuweisen, die Sie zuerst mit einem `Dim` Ausdruck deklarieren müssten.

Die folgende Methode besitzt also Gültigkeit:

```
Sub Square(a As Integer)
a=a*a
MsgBox str(a)
End Sub
```

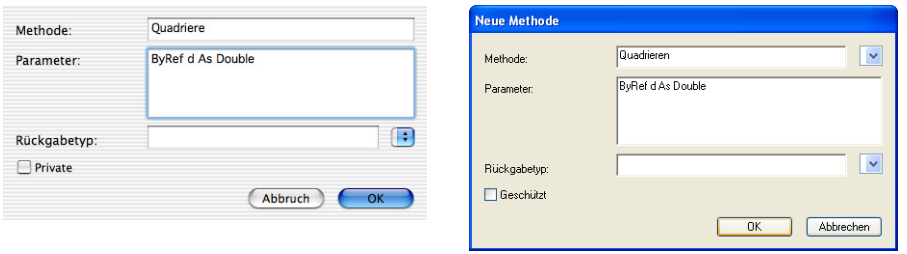
Wenn Sie Ihre eigenen Methoden programmieren, können Sie Informationen auch als Referenz übergeben. In diesem Fall wird ein Zeiger auf das Objekt übergeben, das die Informationen enthält, es wird also keine Kopie der Daten angelegt und Ihre Methode arbeitet mit den Originaldaten. Der Vorteil der Parameterübergabe als Referenz besteht darin, dass die Methode die Werte jedes Parameters verändern kann und diese dann auch in der aufrufenden Methode verän-

dert sind. Das ist nicht möglich ist, wenn die Parameter als Werte übergeben werden, da sie dann ja nur Kopien der Originaldaten sind.

Parameter als Referenz zu übergeben ist vor allem dann von Nutzen, wenn Ihre Methode mehrere Werte zurückliefern muss. Wenn die Parameter als Werte übergeben werden, kann die Methode nur einen einzigen Wert zurückliefern (indem Sie die Methode als function deklarieren und der Wert als Ergebnis der Funktion zurückgegeben wird).

Mit den Schlüsselwörtern ByVal und ByRef können Sie festlegen, welche Art der Parameterübergabe verwendet werden soll. Stillschweigend wird von ByVal ausgegangen, Sie müssen es also nicht extra deklarieren. Um einen Parameter als Referenz zu übergeben, müssen Sie in der Methodendeklaration das Schlüsselwort ByRef verwenden. Das Beispiel in Abbildung 138 zeigt, wie ein Parameter als ByRef deklariert wird.

Abb. 138: Parameterdeklaration ByRef



Falls der Code, der die Methode aufruft, so aussieht:

```
dim a as integer
a=3
quadriere a
editField1.text=str(a)
```

und die Methode selbst so aussieht:

```
a=a*a
```

wird in editField1 die Zahl 9 erscheinen. Wenn Sie nicht explizit ByRef angeben, kann die Methode den an sie übergebenen Wert verändern, aber den geänderten Wert nicht ausgeben.

Sie können auch Arrays als Referenz übergeben. Dazu geben Sie das Array ohne das Schlüsselwort ByRef an. Arrays werden grundsätzlich als Referenz übergeben. Wenn Sie ein String-Array übergeben wollen, das Namen enthält, würden Sie:

```
aNames () as String
```

in den Parameter-Abschnitt im Methoden-Dialog eingeben. Wenn Sie ein Array übergeben, kann die Methode alle Elemente des Arrays ändern.

Vergleichsoperatoren

Es kommt oft vor, dass man zwei Werte vergleichen muss, um herauszufinden, ob eine bestimmte Bedingung erfüllt ist. Tatsächlich stellt man dabei eine Behauptung auf, die entweder wahr oder falsch ist. Die Behauptung „Mein Hund ist eine Katze“ ist normalerweise falsch. Dagegen kann „Mein Hund wiegt mehr als meine Katze“ wahr oder falsch sein. Solche Vergleiche wertet man mit den folgenden Operatoren aus:

Beschreibung	Symbol	Numerisches Beispiel	Ergebnis
Gleichheit	=	5=5	True
Ungleichheit	<>	5<>5	False

Beschreibung	Symbol	Numerisches Beispiel	Ergebnis
Größer als	>	6>5	True
Kleiner als	<	6<5	False
Größer oder gleich	>=	6>=5	True
Kleiner oder gleich	<=	6<=5	False

Auch Strings und boole'sche Werte können für Vergleiche herhalten. String-Vergleiche sind unabhängig von der Groß-/Kleinschreibung (case insensitive) und alphabetisch geordnet. Die Strings „Jannice" und „jannice" sind gleich, aber „Jannice" ist kleiner als „Jason", da „Jannice" im alphabetischen Sinn vor „Jason" kommt. Sollten Sie einen Vergleich benötigen, der die Groß-/Kleinschreibung berücksichtigt oder lexikographisch ist, dann verwenden Sie die StrComp-Funktion. Siehe StrComp in der Sprachreferenz.

Mehrere Vergleiche zugleich

Es ist möglich, mehrere Vergleiche zugleich auszuführen, indem man die Operatoren „or" und „and" verwendet.

And (und) Operator

Verwenden Sie diesen Operator, wenn alle beteiligten Vergleiche „True" ergeben sollen. Wenn im folgenden Beispiel die Variable x den Wert 5 annimmt, so resultiert für den Gesamtausdruck der Wert „False":

```
x>1 And x<5
```

Or (oder) Operator

Ist es ausreichend, dass einer der Komponenten im Vergleich „True" ergibt, dann wird der Operator „or" eingesetzt. Ist x im Beispiel wieder 5, dann ergibt der Vergleich diesmal „True":

```
x>1 Or x<5
```

Not (nicht) Operator

Verwenden Sie den Not Operator, um den Wert eines boole'schen Ausdruck umzukehren. Zum Beispiel prüft

Not (x < 0)

ob x größer oder gleich Null ist. Oder anders ausgedrückt, ob x nicht kleiner Null ist.

Mehrfaches Ausführen von Instruktionen in Schleifen

Manchmal muss eine Reihe von Anweisungen mehrfach ausgeführt werden. Soll beispielsweise nach dem Drücken eines Knopfes dreimal ein Signalton zu hören sein, könnten Sie natürlich einfach dreimal nacheinander das Beep-Kommando aufrufen:

```
Beep
Beep
Beep
```

Sollen Anweisungen aber fünfzigmal ausgeführt werden, oder ist die Anzahl der Ausführungen abhängig von weiteren Bedingungen, so ist dies auf diese Weise entweder sehr ineffizient oder gar nicht möglich. Daher bedienen Sie sich einer Schleife, um diese Anweisungen mehrfach ausführen zu lassen. Davon gibt es verschiedene Arten.

While…Wend

Eine While-Schleife führt die Anweisungen zwischen While und Wend (While End) aus, solange die Bedingung, die bei While angegeben wird, erfüllt ist („True"):

```
Dim n As Integer
While n<10
       n=n+1
       Beep
Wend
```

Die Variable n hat nach dem Anlegen durch Dim den Wert 0. Da 0 kleiner als 10 ist, beginnt die Schleife mit der Ausführung. Wäre im Beispiel n bereits 10, dann würde die Schleife nichts tun und die Programmausführung würde hinter Wend fortgesetzt. Die Anweisungen in der Schleife werden nun immer wieder ausgeführt. Die Variable n wird um eins erhöht und dann wird per Beep ein Signalton ausgegeben. Beim Erreichen von Wend überprüft REALbasic die Bedingung, die bei While steht. Ist diese „True", dann wird das Programm mit der ersten Zeile nach dem While fortgesetzt. In unserem Beispiel wird dann also n wieder um eins erhöht. Hat n den Wert 10 erreicht, dann wird das Programm hinter dem Wend fortgesetzt – die Schleifenausführung ist beendet.

Do…Loop

Do-Loop-Schleifen verhalten sich ähnlich wie While-Schleifen. Im Unterschied zu diesen werden Do-Loop-Schleifen solange ausgeführt, bis eine Bedingung „True" ist, während While-Schleifen solange ausgeführt werden, solange eine Bedingung „True" bleibt. Do-Loop-Schleifen sind flexibler, da man die Bedingung am Anfang oder am Ende der Schleife prüfen kann:

```
Do Until n=10
 n=n+1
 Beep
Loop
```

oder

```
Do
 n=n+1
 Beep
Loop Until n=10
```

Der Unterschied liegt darin, dass die erste Schleife nicht ausgeführt wird, wenn n schon den Wert 10 hat, während die zweite Schleife unabhängig vom Wert von n mindestens einmal ausgeführt wird, da erst am Ende abgefragt wird, welchen Wert n hat.

Es ist möglich, eine Do-Loop-Schleife zu programmieren, die keine Bedingungen abfragt:

```
Do
 n=n+1
 Beep
Loop
```

Weil hier keine Endbedingung vorliegt, wird die Schleife endlos ausgeführt. Mit der Exit-Methode lässt sich die Ausführung einer solchen Schleife beenden. Allerdings ist dies sehr schlechter Programmierstil, denn man muss erstmal die gesamte Schleife durchforsten, bis man herausfindet, wodurch sie beendet wird.

Endlosschleifen

Sie müssen immer sicherstellen, dass die Endbedingungen Ihrer Schleifen auch irgendwann eintreffen. Ist dies nicht der Fall, dann bleibt das Programm an dieser Stelle hängen. In der Praxis wird Ihnen das während der Entwicklung passieren. Ein Programm, das in einer Schleife steckenbleibt, lässt sich mit ⌘-Shift-„." abbrechen. Mit ⌘-K (Windows: Strg-K oder Strg+Alt+Entfernen) lässt sich das Programm, das in der Schleife steckt, dann beenden.

Lange Schleifen

Wenn eine Schleife begonnen wird, übernimmt ihr Prozess die Kontrolle und erlaubt es dem Anwender nicht, Interface-Elemente wie Menüs, Knöpfe und Scrollbalken zu bedienen. Auf schnellen Rechnern und kleinen Schleifen ist das kein größeres Problem, da die Schleife meist schneller abgearbeitet ist als der Anwender die nächste Aktion unternehmen will. Wenn das jedoch nicht der Fall ist, können Sie folgendes machen:

- Wenn der Anwender warten soll, bis die Schleife beendet ist, bevor er etwas anderes machen darf (z.B., weil eine Aktion des Anwenders das Ergebnis der Schleife ungültig machen könnte), können Sie anzeigen, dass eine längere Operation im Gange ist, indem Sie den Mauszeiger in die Armbanduhr verwandeln, bis die Schleife beendet ist. Bedenken Sie dabei, dass der Anwender in diesem Fall die Schleife nicht vorzeitig abbrechen kann. Mehr Informationen finden Sie im Abschnitt über MouseCursor in der Sprachreferenz.
- Wenn es dem Anwender erlaubt sein soll, andere Dinge zu tun, während die Schleife läuft, sollten Sie den Code der Schleife in einen separaten Thread platzieren. Ein Thread läuft als Hintergrund-Task und erlaubt dem Anwender im Vordergrund weiterzuarbeiten.

Vermeiden Sie lang andauernde Operationen innerhalb von Schleifen. Versuchen Sie, diese einmalig außerhalb vorzunehmen und dann darauf aus der Schleife heraus zu verweisen. Wenn sich dies nicht vermeiden lässt, können Sie die Methode DoEvents der Application Klasse verwenden, um Prozessorzeit an REALbasic zurückzugeben, damit ausstehende Events abgearbeitet werden können. Dies betrifft zum Beispiel Tastatur- oder Mauseingaben. Näheres finden Sie in Eintrag über die Application Klasse in der Sprachreferenz.

For…Next

While- und Do-Loop-Schleifen sind perfekt geeignet, wenn die Anzahl der Ausführungen von komplexen Bedingungen abhängt. Kann die Anzahl der Schleifendurchgänge numerisch bestimmt werden, dann verwendet man gewöhnlich eine For-Next-Schleife. Sollen zum Beispiel in einer ListBox die Zahlen eins bis zehn eingetragen werden, dann bietet sich die Verwendung einer For-Next-Schleife an, da die Anzahl der Ausführungen bereits feststeht. Eine solche Schleife ist wie folgt aufgebaut:

```
Dim zähler, Startwert, Endwert As Integer
.
.
For zähler=Startwert to Endwert
 [eigener Programmcode]
Next
```

Im Beispiel ist der Zähler der For-Next-Schleife als Integer deklariert. Dies ist zwar üblich, aber nicht zwingend erforderlich. Sie können die Zähler-Variable auch als Single oder Double deklarieren. Bei Beginn der Schleifenausführung wird die Zählvariable auf den Startwert gesetzt und beim Erreichen von Next automatisch um eins erhöht. Hat sie den Endwert erreicht, wird die Schleife noch einmal ausgeführt. Erst wenn der Zähler größer ist als der Endwert, endet die Schleifenausführung. Unser Beispiel sieht so aus:

```
Dim i As Integer
For i=1 to 10
 ListBox1.AddRow Str(i)
Next
```

In der Schleife wird jeweils die Zahl i in eine Zeichenkette gewandelt und an die ListBox übergeben. Dies geschieht genau zehnmal.

Anmerkung: Der Buchstabe „i“ wird sehr häufig als Zähler für Schleifen verwendet. Dies geht auf die Programmierung in FORTRAN zurück, denn in FORTRAN sind die Buchstaben von I bis N automatisch als Integer-Variablen deklariert. Daher wurden diese Buchstaben normalerweise für Zähler verwendet. Sind Schleifen ineinander verschachtelt, dann werden

die nächsten Buchstaben verwendet, also J, K und so weiter. Wenn man sich daran hält, dann macht man es anderen Programmierern leichter, den Code zu durchschauen.

Normalerweise werden die Schleifenzähler um eins erhöht. Dies lässt sich mit dem Befehl Step ändern. Im Beispiel wird der Zähler immer um fünf erhöht:

```
Dim i As Integer
For i=5 to 100 Step 5
 ListBox1.AddRow Str(i)
Next
```

Im nächsten Beispiel beginnt der Zähler bei 100 und wird jeweils um fünf erniedrigt:

```
Dim i As Integer
For i=100 Downto 1 Step 5
 ListBox1.AddRow Str(i)
Next
```

Bisher haben wir Fälle behandelt, in denen *Startwert* und *Endwert* Integer-Zahlen waren. Wenn entweder *Startwert* oder *Endwert* Ausdrücke sind, die berechnet werden müssen, führt die For-Schleife diese Berechnung jedesmal durch, wenn der Zähler erhöht wird – auch wenn die Berechnung jedesmal zum gleichen Ergebnis führt.

Daher ist es ratsam, die Berechnungen durchzuführen, bevor die Schleife beginnt. Wenn Sie z.B. eine Schleife benötigen, die alle installierten Fonts bearbeitet, können Sie im Voraus nicht wissen, wieviele das sind. Mit der Funktion FontCount können Sie die Anzahl der Fonts ermitteln. Wenn Sie die Schleife folgendermaßen verwenden:

```
For i=0 to FontCount-1
.
.
Next
```

wird die Schleife sehr viel langsamer ablaufen, als wenn Sie den Wert nur einmal ermitteln:

```
Dim nFonts as integer
nFonts=FontCount-1
For i=0 to nFonts
.
.
Next
```

Der Geschwindigkeitsunterschied ist meist nicht zu bemerken, bis es sich um eine sehr lange Schleife handelt. Auf dem Rechner, auf dem dieses Handbuch geschrieben wurde und auf dem 120 Fonts installiert sind, ist der Unterschied zwischen beiden Varianten gerade einmal 1/250 Sekunde.

Schleifen lassen sich ineinander schachteln. Man muss nur darauf achten, dass die Zählvariablen unterschiedlich sind und sich nicht gegenseitig verändern. Im folgenden Beispiel soll gezählt werden, in wievielen Zellen einer mehrspaltigen ListBox das Wort „hello“ steht:

```
Dim row, column, count As Integer
For row=0 to listBox1.ListCount-1
 For column=0 to listBox1.ColumnCount-1
 if listbox1.cell(row,column)="hello" then
 count=count+1
 End if
 Next
Next
MsgBox Str(count)
```

Eine andere Vorgehensweise ist die Verwendung von Namen, wie sie von FORTRAN-Programmierern zuerst benutzt wurde. Hierbei verwendet man die Buchstaben des Alphabets, beginnend mit i als Zähler. Auf diese Weise sieht man sofort, welche Schleife sich in welcher befindet, ohne dass Sie überlegen müssen, was die Schleifen machen.

For-Next-Schleifen sind anderen Schleifen vorzuziehen, weil der daraus resultierende Maschinencode effizienter ist, da sich solche Schleifen besser compilieren lassen.

For...Each

For…Each durchläuft die Schleife nicht für jeden Wert eines Zählers, sondern für jedes Element eines Arrays:

```
Function SumStuff(values() as Double) as Double
Dim sum, element as Double
For Each element In values
sum=sum+element
Next
Return Sum
```

„values“ ist ein eindimensionales Array mit Zahlen, das an die Funktion „SumStuff“ übergeben wird. Im For..Each-Ausdruck übernimmt die Variable „element“, die auf ein Element im Array verweist, die Rolle der Zählervariable. Die For...Each-Schleife führt für jedes Element im Array die Anweisungen zwischen dem For..Each-Ausdruck und dem Next-Ausdruck aus.

Da das Array nicht unbedingt aus Zahlen bestehen muss, können Sie mit diesem Ausdruck Objekte beliebigen Typs verarbeiten. Dabei könnte es sich um Bilder, Farben, Dokumente, Datensätze usw. handeln.

Um die Funktion aus dem Beispiel aufzurufen, übergeben Sie ihr als Parameter ein Array aus Double-Werten:

```
s=SumStuff(MyNumbers)
```

Die Variable s wurde dabei als Double deklariert.

Verzweigen

Der Programmcode in Ihren Methoden wird von oben nach unten Zeile für Zeile ausgeführt. Oft ist es aber erforderlich, dass abhängig von bestimmten Bedingungen nur Teile des Codes einer Methode ausgeführt werden sollen. Dazu verzweigt man entweder über If…Then…End If oder Select…Case.

If…Then…End If

Mit If…Then überprüft man eine logische Bedingung und führt den darin eingeschlossenen Code abhängig vom Ergebnis aus oder überspringt ihn. Eine solche Anweisung sieht so aus:

```
If Bedingung Then
 [eigener Programmcode]
End If
```

Angenommen, Sie haben eine Integer-Variable namens Monat und möchten nur dann etwas bestimmtes ausführen, wenn diese den Wert 1 hat, dann tun Sie dies:

```
If Monat=1 Then
 [eigener Programmcode]
End If
```

Monat=1 ist ein boole'scher Ausdruck, der entweder True oder False annimmt, je nachdem ob Monat den Wert 1 hat oder nicht.

Nehmen wir mal an, dass Sie eine Checkbox verwenden, die darüber bestimmen soll, ob ein bestimmter Programmteil ausgeführt wird. Sie überprüfen dann den boole'schen Wert der Checkbox ganz einfach so:

```
If checkbox1.value Then
 [eigener Programmcode]
End If
```

If…Then…Else…End If

Es kommt auch vor, dass man zwei unterschiedliche Dinge tun möchte. Die eine Aktion soll stattfinden, wenn eine Bedingung erfüllt ist und ist das nicht der Fall, dann soll etwas anderes geschehen. Man unterteilt den Code dann in zwei Teile. Diese werden von Else getrennt:

```
If month=1 Then
 MsgBox "Es ist Januar."
Else
 MsgBox "Es ist nicht Januar."
End If
```

If…Then…ElseIf…End If

Soll im Else-Teil noch eine weitere Bedingung geprüft werden, kann man dafür ElseIf verwenden:

```
If month=1 Then
 MsgBox "Es ist Januar."
ElseIf month<4 Then
 MsgBox "Es ist zwar nicht Januar, aber immer noch Winter."
End If
```

Dies hätte man natürlich auch bewirken können, indem man einen weiteren If…Then-Teil innerhalb des Else-Teils eingebaut hätte. Das würde aber ein weiteres End If ins Spiel bringen und das Programm wäre schwerer zu lesen. Der wahre Beitrag von ElseIf zur Übersichtlichkeit des Programms wird klar, wenn man sich folgendes Beispiel ansieht, das zeigt, dass mehrere ElseIf-Abfragen hintereinander stehen dürfen:

```
If month=1 Then
 MsgBox "Es ist Januar."
ElseIf month<4 Then
 MsgBox "Es ist noch Winter."
ElseIf month<6 Then
 MsgBox "Jetzt ist es Frühling."
End If
```

Ist die If-Bedingung False, dann testet REALbasic der Reihe nach alle ElseIf-Bedingungen. Sobald eine davon True ergibt, wird der dazugehörende Code ausgeführt und das Programm nach EndIf fortgesetzt. Beachten Sie, dass weitere ElseIf-Teile danach nicht mehr zur Ausführung kommen.

Select…Case

Wenn Sie mehrere mögliche Werte einer Variablen oder einer Eigenschaft zur Verzweigung verwenden möchten, dann bietet sich die Verwendung von Select…Case an:

```
If Tag=2 Then
 MsgBox "Montag."
ElseIf Tag=3 Then
 MsgBox "Dienstag."
ElseIf Tag=4 Then
 MsgBox "Mittwoch."
ElseIf Tag=5 Then
 MsgBox "Donnerstag."
ElseIf Tag=6 Then
 MsgBox "Freitag."
Else
 MsgBox "Wochenende."
End If
```

Diese Bedingungen schließen sich alle gegenseitig aus. Es ist also immer nur jeweils eine erfüllt. In einem Select…Case ist das allerdings erheblich übersichtlicher:

```
Select Case Tag
Case 2
 MsgBox "Montag."
Case 3
 MsgBox "Dienstag."
Case 4
 MsgBox "Mittwoch."
Case 5
 MsgBox "Donnerstag."
Case 6
 MsgBox "Freitag."
Else
 MsgBox "Wochenende."
End Select
```

Die Select…Case-Anweisung vergleicht die hinter Select Case angegebene Variable oder Eigenschaft nacheinander mit den in den Case-Zweigen aufgeführten Werten. Trifft ein Wert zu, wird der Code des betreffenden Case-Zweiges bis zum nächsten Case ausgeführt. Treffen mehrere Case-Zweige zu, wird trotzdem nur der erste zutreffende Zweig ausgeführt. Zu einer Select…Case-Anweisung kann ein abschließender Else-Zweig gehören, der für alle nicht explizit angegebenen Case-Werte ausgeführt wird.

Die Select…Case-Anweisung funktioniert mit Variablen beliebigen Typs, wie z.B. String, Integer, Single, Double, Boolean und Color. Zum Beispiel können Sie Farben vergleichen:

```
Dim c as Color
c=&cFF0000 //Rot
Select case c
 case &c00FF00 // Grün
  MsgBox "Grün"
 case &cFF0000 // Rot
  MsgBox "Rot"
 case &c0000FF // Blau
  MsgBox "Blau"
End select
```

Ein Case-Ausdruck akzeptiert auch mehrere Werte, die durch Komma getrennt sind. Ein Beispiel:

```
Dim c as color
c=&cFF0000 //Rot
Select case c
 case &c00FF00,&cFF0000 //Grün,Rot
  msgBox "Grün oder Rot"
 case &cFF0000 //Rot
  MsgBox "Rot"
 case &c0000FF //Blau
  MsgBox "Blau"
End select
```

Im letzten Beispiel ist der erste Case-Ausdruck wahr, da c den Wert &cFF0000 hat. Obwohl der zweite Case-Ausdruck ebenfalls wahr wäre, wird dieser nicht ausgeführt, da er nicht der erste zutreffende Ausdruck war.

Ein Case-Ausdruck akzeptiert sogar einen Wertebereich, den Sie mit dem „To“ Schlüsselwort angeben. Zum Beispiel:

```
Dim i as Integer = 53
Select case i
 Case 1 to 25
  msgBox "25 or less"
```

```
 Case 26 to 50
  MsgBox "26 to 50"
 Case 51 to 100
  MsgBox "51 to 100"
End Select
```

Im Beispiel wäre der dritte Case-Ausdruck wahr.

Sie können auch Wertebereiche mit einzelnen Werten kombinieren:

```
Case 0, 26 to 50, 75, 100 to 200
```

Sie können auch auf Ungleichheit prüfen, indem Sie das „Is"-Schlüsselwort und einen Vergleichsoperator verwenden (z.B. <, >, <=, >=). Zum Beispiel:

```
Dim i as Integer = 10
Select Case i
 Case Is <= 10
//this case selected
 Case Is > 10
//this case not selected
End Select
```

Auch Vergleichsoperatoren lassen sich mit einfachen Werten kombinieren:

```
Dim i as Integer = 75
Select Case i
Case 0, Is <=10,100
//case not selected
Case Is > 10, Is < 99
//case selected
End Select
```

Sogar Funktionen lassen sich verwenden, wenn sie den erforderlichen Datentyp zurückgeben:

```
Dim i as Integer = 4
Dim a as Integer = 2
Select Case i
 Case Functx(a)
  //case 1
 Case a
  //case 2
Else
  //no match
End Select
```

Die Funktion im ersten Case-Ausdruck sähe so aus:

```
Function Functx(a As Integer) as Integer
 Return a*a
End Function
```

Die Funktion quadriert den angegeben Wert, so dass der erste Case-Ausdruck zutrifft.

Im Falle einer derart einfachen Funktion können Sie die Berechnung auch direkt in den Case-Ausdruck schreiben. Damit würde dieser so aussehen:

```
Case a*a
```

Die Select…Case-Anweisung kann sogar Variablen vom Typ Objekt vergleichen. Im folgenden Beispiel wird Select…Case verwendet, um den Button zu ermitteln, den der Benutzer in einem MessageDialog gedrückt hat. Dabei werden Objekte vom Typ MessageDialogButton verglichen, um zu ermitteln, welcher der drei Buttons gedrückt wurde.

```
Dim d as New MessageDialog //MessageDialog-Objekt deklarieren
Dim b as MessageDialogButton //Das zu prüfende Ergebnis nach dem Klick
```

```
d.icon=1 //Zeige ein 'Warnung'-Icons
d.ActionButton.Caption="Sichern"
d.CancelButton.Visible=True //Abbrechen-Button anzeigen
d.AlternateActionButton.Visible=True //Alternativ-Button anzeigen
d.AlternateActionCaption='Nicht sichern"
d.Message="Änderungen vor Beenden sichern?"
d.Explanation="Die Änderungen gehen verloren, wenn Sie sie nicht sichern."
b=d.ShowModal //Zeige Dialog modal an
Select Case b   //Der MessageDialogButton der d zugewiesen wurde
 Case d.ActionButton    //ermittle welcher Button gedrückt wurde
 //Benutzer hat sichern geklickt
 Case d.AlternateActionButton
 //Benutzer hat nicht sichern geklickt
 Case d.CancelButton
 //Benutzer hat abbrechen geklickt
End Select
```

Kapitel 5 Programmieren mit Events und Objekten

Die meisten Teile Ihres Programms werden auf Eingaben des Benutzers reagieren. Der Benutzer löst dabei Events, also Ereignisse aus, auf die Ihr Programm reagieren muss. Sie müssen verstehen, wie Events funktionieren, wann sie auftreten und wie man sie zur Programmsteuerung verwendet, damit das Programm letzten Endes das tut, was Sie möchten.

In diesem Kapitel werden Sie einiges über Event-gesteuerte Programmierung lernen, wie man den Code-Editor verwendet und wie man dafür sorgt, dass ein Programm auf Eingaben des Benutzers reagiert.

Inhalt

- Event-gesteuertes Programmieren
- Benutzung des Code Editors
- Drucken und Export/Import von Programmcode
- Programmierung von Event-Handlern

Eventgesteuertes Programmieren

Die Benutzer Ihres Programms werden mit dem Programm kommunizieren, indem sie mit der Maus klicken oder etwas auf der Tastatur eingeben. Jedesmal, wenn ein Benutzer mit der Maus auf das Benutzer-Interface Ihres Programmes klickt oder beispielsweise mit der Tastatur etwas in ein EditField eingibt, wird ein Event ausgelöst. Events sagen Ihrer Anwendung ganz einfach, was der Benutzer getan hat und wo dies passiert ist. Es kann auch sein, dass ein Event indirekt andere Events auslöst. Klickt der Benutzer auf einen Menüpunkt, dann ist dies ein Event. Öffnet sich daraufhin ein Fenster, dann wird durch das Event „Menüpunkt ausführen“ indirekt ein zweites Event, nämlich das Öffnen des Fensters, ausgelöst.

Jedes REALbasic-Objekt kann den zur Behandlung der empfangenen Events benötigten Code enthalten.

Ein PushButton kann beispielsweise den Code enthalten, der ausgeführt werden soll, wenn das Event eintritt, das der Benutzer auf ihn klickt. Ein Objekt kann sogar auf Events reagieren, an die Sie vielleicht noch gar nicht gedacht haben. Zum Beispiel kann ein Event auftreten, wenn der Benutzer einfach nur die Maus über das Objekt bewegt. Wenn der Benutzer ein Event auslöst, dann prüft REALbasic, ob bei dem betreffenden Objekt Code zur Behandlung dieses Events vorgesehen ist. Sollte das der Fall sein, wird der entsprechende Code ausgeführt und danach wartet REALbasic auf das nächste Event. Dies wird fortgesetzt, bis schließlich ein Event eintritt, das die Programmausführung beendet (normalerweise wenn der Benutzer Beenden im Ablage-Menü auswählt).

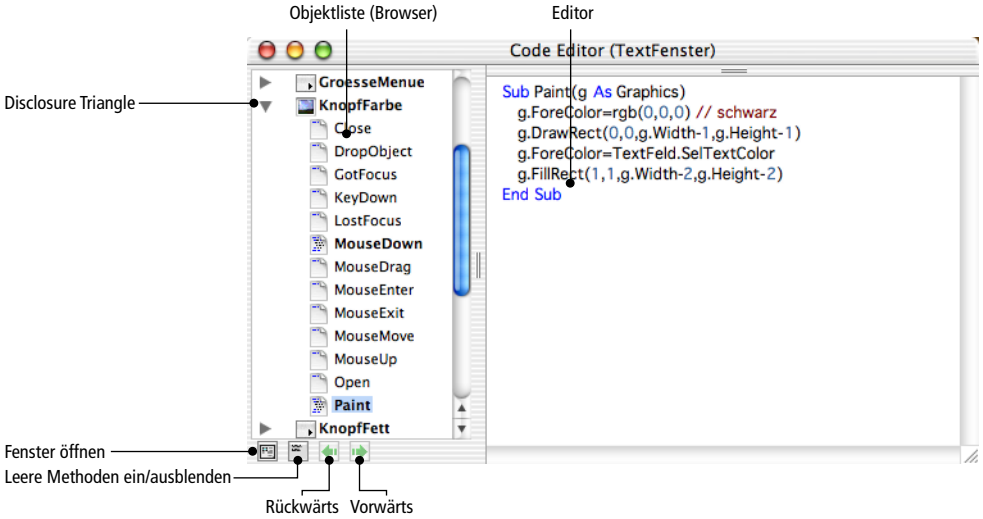
PushButtons haben unter anderem ein Event, das Action heißt und ausgelöst wird, wenn der Benutzer auf den Knopf klickt. Den Programmcode, der ausgeführt wird, wenn das Event ausgelöst wird, nennt man Event-Handler, eben weil hier der Code abgearbeitet wird, der zu dem ausgelösten Event gehört. Soll der Knopf zum Öffnen eines Fensters dienen, dann löst er indirekt wieder ein weiteres Event aus, nämlich das Open-Event des neuen Fensters, das immer automatisch beim Öffnen eines Fensters ausgelöst wird.

Es gibt eine ganze Menge von Events, die jedes Objekt, das Sie verwenden, empfangen kann. Glücklicherweise müssen Sie nicht alle Events selbst behandeln.

Benutzung des Code-Editors

Der Code-Editor dient zur Eingabe des Programmcodes der einzelnen Events, die für die verwendeten Objekte Ihres Benutzer-Interface auftreten können. Hier werden außerdem neue Methoden und Eigenschaften angelegt. Der Code-Editor besteht aus dem eigentlichen Editor und einem Browser-Fenster.

Abb. 139: Der Code Editor



Der Browser zeigt eine hierarchische Liste der Komponenten, aus denen ein bestimmtes Fenster besteht. Er zeigt folgende Elemente, die für das Fenster verwendet wurden:

- Steuerelemente (Controls)
- Events
- Menü-Handler (Menu Handlers)
- Methoden (Methods)
- Eigenschaften (Properties)
- Notizen

Öffnen des Code-Editors

Der Code-Editor wird dazu verwendet, um Code von Steuerelementen, Fenstern, Klassen und Modulen zu editieren. Sie können den Code-Editor auf verschiedene Arten öffnen.

Um den Code-Editor für ein Fenster aus dem Projektfenster zu öffnen, unternehmen Sie folgendes:

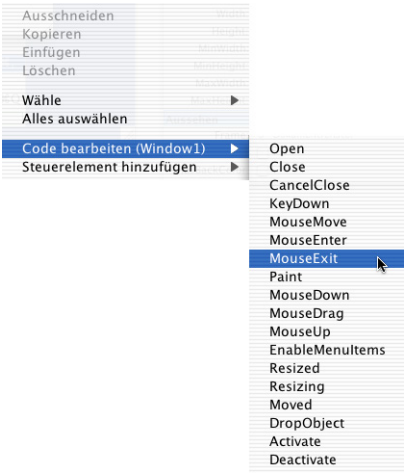
1. Rufen Sie das Kontextmenü des Fensters auf, indem Sie die Ctrl-Taste gedrückt halten und gleichzeitig auf den Namen des Fensters klicken. Unter Windows klicken Sie mit der rechten Maustaste auf den Namen des Fensters im Projektfenster.
2. Wählen Sie **Code bearbeiten** aus dem Kontextmenü.

Um den Code-Editor für ein Fenster oder ein Steuerelement von einem Fenstereditor aus auszurufen, unternehmen Sie folgendes:

Klicken Sie auf das Fenster im Projektfenster und drücken Sie Strg+Enter oder Alt+Tab (Macintosh) oder führen Sie einen Ctrl-Klick (Windows: Rechtsklick) auf dem Fenster aus, um das Kontextmenü des Fensters anzuzeigen. Wählen Sie

dann aus dem **Code bearbeiten**-Menü den Event, den Sie öffnen möchten. Wenn Sie einen Doppelklick auf einem Fenster ausführen, öffnet sich im Code-Editor der Open-Event-Handler des Fensters. Mit dem Kontextmenü können Sie auswählen, welchen Event-Handler Sie öffnen möchten. Das **Code bearbeiten**-Untermenü enthält eine komplette Liste der Events für das Fenster.

Abb. 140: Das „Code bearbeiten"-Untermenü eines Fensters



Um für ein Steuerelement den Default-Event im Code-Editor zu öffnen, gehen Sie folgendermaßen vor:

1. Falls es nicht schon offen ist, öffnen Sie das Fenster, welches das Steuerelement enthält.
2. Führen Sie auf dem Steuerelement einen Doppelklick aus oder wählen Sie das Steuerelement aus und drücken Sie die Return-Taste.

Dies öffnet den Code-Editor für das Elternfenster des Steuerelements. Der Browser des Code-Editors expandiert dann automatisch die Kategorie „Steuerelemente", in dieser das Steuerelement, das Sie angeklickt haben und selektiert den Default-Eventhandler (z.B. bei einem PushButton den Action-Eventhandler).

Jedes Steuerelement besitzt seinen eigenen Default-Eventhandler. Der Default-Event eines PushButtons ist beispielsweise der Action-Event, der eines Scrollbars der ValueChanged-Event und der einer ListBox der Change-Event.

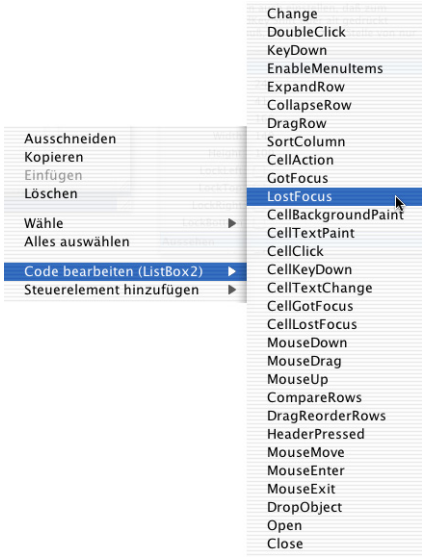
Wenn Sie einen Doppelklick auf ein Steuerelement ausführen, öffnet sich der Code-Editor und zeigt den Default-Event-Handler für dieses Steuerelement.

Um für ein bestimmtes Steuerelement gezielt einen Event im Code-Editor zu öffnen, gehen Sie folgendermaßen vor:

1. Führen Sie einen Ctrl-Klick (Windows: Rechtsklick) auf ein Steuerelement im Fenster aus, um das Kontextmenü für dieses Steuerelement anzuzeigen.
2. Wählen Sie aus dem **Code bearbeiten**-Untermenü den Event-Handler aus, den Sie bearbeiten wollen.

Der **Code bearbeiten**-Menüeintrag für ein individuelles Steuerelement zeigt die Events für dieses Steuerelement an. Verwenden Sie es dazu, um einen bestimmten Event-Handler direkt anzusteuern.

Abb. 141: Das „Code bearbeiten"-Untermenü eines ListBox-Steuerelements

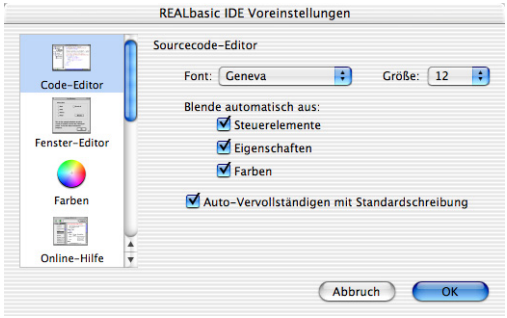


Um den Code-Editor eines Moduls oder einer Klasse zu öffnen, führen Sie einen Doppelklick auf dem Namen des Moduls oder der Klasse im Projektfenster aus. Sie können auch **Bearbeiten** aus dem Kontextmenü des Moduls oder der Klasse wählen. Module sind eigenständige Objekte, in denen Sie globale Konstanten, globale Methoden und globale Eigenschaften verwalten können. Sie können auf die Konstanten, Methoden und Eigenschaften eines Moduls von jeder Stelle der Applikation aus zugreifen. Klassen sind wiederverwendbare Objekte, die auf bestehenden Objekten in REALbasic basieren. Weitere Informationen finden Sie in den Kapiteln „Globale Funktionen durch Module" auf Seite 229 und „Wiederverwendbare Objekte durch Klassen" auf Seite 304.

Konfigurieren des Code-Editors

Sie können verschiedene Voreinstellungen für den Code-Editor vornehmen. Öffnen Sie dazu unter Windows und Mac OS Classic mit „Bearbeiten/Einstellungen" (oder unter Mac OS X mit „REALbasic/Einstellungen") die in Abbildung 142 gezeigte Dialogbox. Die Voreinstellungen für den Code-Editor sind auf zwei Dialogseiten verteilt: **Code-Editor** und **Farben**.

Abb. 142: Der Einstellungen-Dialog

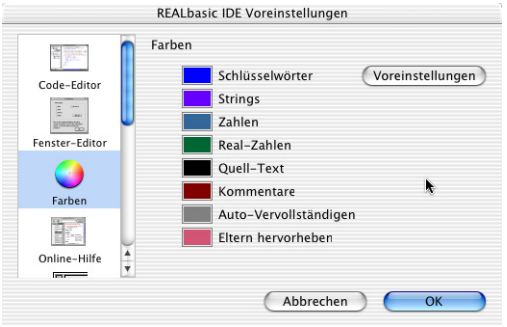


Innerhalb der Code-Editor-Einstellungen können Sie folgende Einstellungen vornehmen:

- **Font:** Legt die Schriftart und -größe für den Code-Editor fest. Für den Ausdruck können Sie getrennte Einstellungen vornehmen. Wählen Sie dazu das Menü **Drucken**.
- **Automatisches Ausblenden von Steuerelementepalette, Eigenschaftenfenster und Farbenfenster:** Gemäß Voreinstellung blendet REALbasic Steuerelementepalette, Eigenschaftenfenster und Farbenfenster aus, wenn der Code-Editor aktiv ist. Für jedes der genannten Elemente können Sie diese Einstellung zurücknehmen. Wenn Sie sich, während Sie im Code-Editor arbeiten, die Steuerelementepalette oder das Farbenfenster anzeigen lassen, können Sie Ihrem Code per Drag & Drop Farben oder Steuerelemente hinzufügen. Wenn Sie eine Farbe aus dem Farbenfenster in den Code-Editor draggen, wird der entsprechende RGB-Wert in den Code eingefügt. Wenn Sie ein Steuerelement aus der Steuerelementepalette in den Code-Editor draggen, wird wird dessen Name in den Quelltext übernommen.
- **Auto-Vervollständigen mit Standardschreibung:** Wenn Sie diese Einstellung wählen, verwendet REALbasic zur Auto-Vervollständigung die per Konvention festgelegte Groß/Kleinschreibung für Schlüsselwörter, wie sie in der Sprachreferenz vorgegeben ist. Ist diese Einstellung nicht ausgewählt, belässt es REALbasic bei der von Ihnen getippten Schreibweise.

Auf der Dialogseite **Farben** können Sie die Farben für den Code-Editor festlegen.

Abb. 143: Die Rubrik „Farben" in den REALbasic-Einstellungen



Sie können folgenden Elementen Farbe zuweisen:

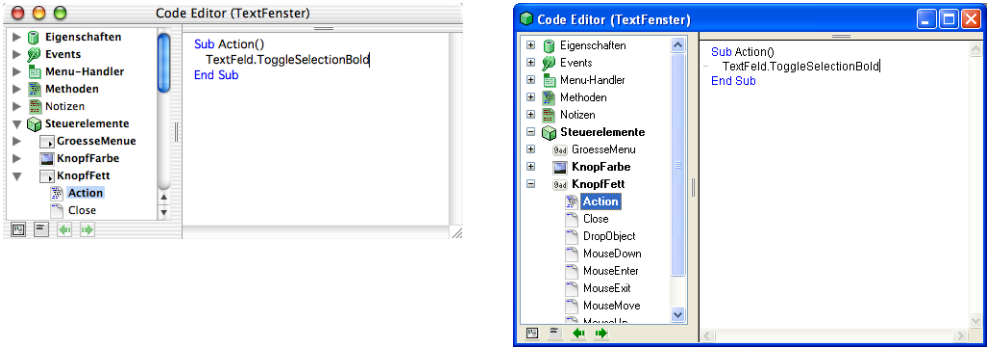
- **Schlüsselwörtern:** REALbasic-Sprachelementen, wie zum Beispiel Kontrollstrukturen (If, While, etc.) und Datentypen (Integer, Color, Double, etc.).
- **Strings:** Strings, die Sie im Code einsetzen, wie beispielsweise Text, den Sie an eine MsgBox-Funktion übergeben, um dem Benutzer einen Warnhinweis anzuzeigen.
- **Zahlen:** Integer-Zahlen.
- **Real-Zahlen:** Zahlen mit Dezimalstellen. Die Unterscheidung zwischen Real-Zahlen und Zahlen ist hier unabhängig vom deklarierten Datentyp. Wenn Sie beispielsweise die Variable i als Integer deklarieren und ihr anschließend den Wert i=5,76 zuweisen, dann greift REALbasic für „5,76" auf die Einstellungen für die Real-Zahlen zurück.
- **Quellcode:** Variablen, Steuerelemente, Klassen, Operatoren, Methoden, Eigenschaften usw.
- **Kommentaren:** Kommentar-Text. Kommentaren werden zwei Schrägstriche(//), das Schlüsselwort REM oder ein einfaches Anführungszeichen(') vorangestellt.
- **Text, der automatisch vervollständigt wurde:** Text, der durch die Auto-Vervollständigen-Option erzeugt wurde.
- **Eltern hervorheben:** Wenn Sie die Vorteile der Steuerelemente-Hierarchie nutzen, hebt REALbasic das Eltern-Steuerelement hervor, wenn Sie ihm ein untergeordnetes Steuerelement (Kind) zuordnen.

Um eine Farbe zu ändern, klicken Sie auf ein Farbfeld. Die Farbauswahl erscheint. Wählen Sie eine neue Farbe und bestätigen mit **OK**. Um alle Farbeinstellungen wieder in den Ausgangszustand zu versetzen, klicken Sie auf den **Voreinstellungen**-Button.

Der Browser

In der oberen Hierarchieebene werden die Namen der einzelnen Kategorien gezeigt, also Steuerelemente, Methoden, Eigenschaften etc. Klicken Sie auf das Dreieck vor einem Kategorienamen, um die Objekte der Kategorie anzuzeigen. Auf diese Weise können Sie wiederum die zu dem Objekt gehörenden Events etc. aufklappen. In folgender Abbildung können Sie sehen, dass das Fenster „TextFenster“ einen BevelButton namens KnopfFett hat:

Abb. 144: Der Action-Event-Handler eines BevelButtons



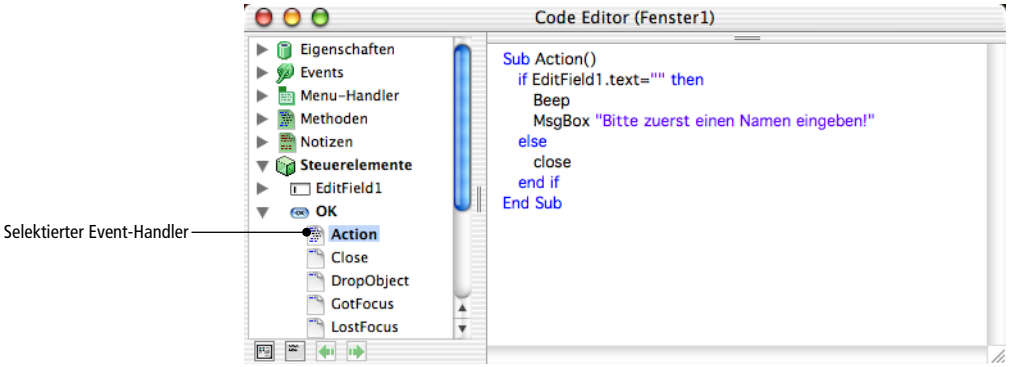
Dieser BevelButton kennt folgende Event-Handler:

- Action
- Close
- DropObject
- MouseDown
- MouseEnter
- MouseExit
- MouseMove
- MouseUp
- Open

Event-Handler sind innerhalb eines Objekts oder eines Steuerelements alphabetisch aufgelistet. Wenn Sie zum ersten Mal die Event-Handler für ein Steuerelement öffnen (zum Beispiel, indem Sie in einem Fenster auf ein Steuerelement doppelklicken), ist der voreingestellte Event-Handler ausgewählt. Es kann sich dabei um den ersten Event-Handler in der Liste handeln (zum Beispiel der Action-Event-Handler bei einem PushButton), aber das ist nicht immer der Fall. Für das Canvas-Steuerelement beispielsweise ist der Paint-Event-Handler der automatisch ausgewählte Event-Handler.

Wenn Sie in der Objektliste auf den Event-Handler eines Steuerelements klicken, wird im Code-Editor der dazugehörige Quelltext angezeigt.

Abb. 145: Code eines Event-Handlers



Existiert zu einem Browser-Eintrag bereits Programmcode, werden der Browser-Eintrag und alle übergeordneten Einträge fett hervorgehoben. In der obigen Abbildung enthält der Action-Event des OK-Knopfes Programmcode. Deshalb wird der Name des Event-Handlers, der Name des Steuerelements und dessen Kategorie fett dargestellt. So kann man schnell herausfinden, wo Codestücke untergebracht sind, ohne manuell alle Einträge in der Liste anklicken zu müssen.

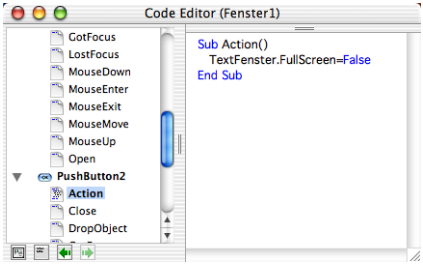
Leere Methoden anzeigen und verstecken

Am unteren Rand der Objektliste gibt es vier Navigationsknöpfe, mit denen man die gängigsten Operationen im Code-Editor auslösen kann. Der zweite Knopf zeigt und versteckt leere Methoden – also Methoden, für die kein Programmcode existiert. Folgende Abbildung zeigt den Knopf in beiden Zuständen:

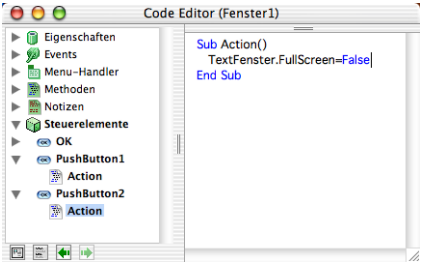
Abb. 146: Die Knöpfe zum Zeigen und Verstecken von Methoden

Wenn leere Methoden versteckt sind, können mehr Objekte angezeigt werden. Folgende Abbildung zeigt beide Zustände im Vergleich:

Abb. 147: Der Code-Editor mit eingeblendeten und versteckten leeren Methoden



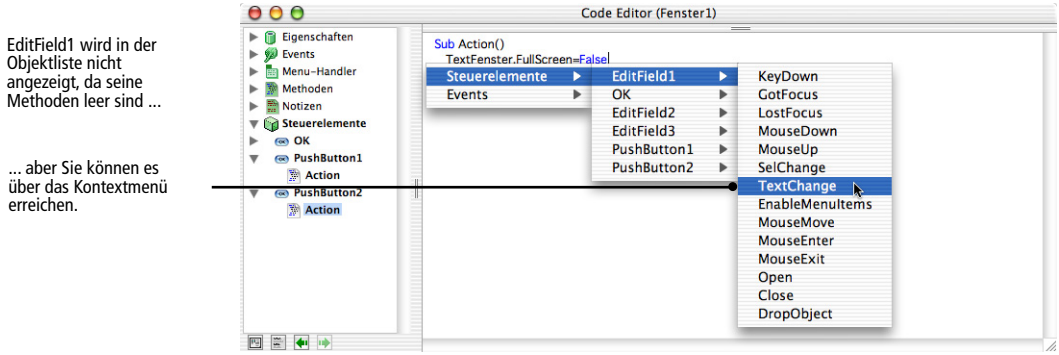
Leere Methoden werden angezeigt
Sie müssen in der Objektliste scrollen, um alle Methoden anschauen zu können. Sie können dadurch jedoch einer leeren Methode Code hinzufügen, indem Sie einfach auf diese klicken.



Leere Methoden werden nicht angezeigt
Sie haben Zugriff auf alle Objekte und Events, die Code besitzen, ohne dass Sie in der Objektliste scrollen müssen. Sie können aber auch kein Objekt expandieren, das keinen Code besitzt. Sie müssen, um einer leeren Methode Code hinzuzufügen, zuerst in den Modus umschalten, dass leere Methoden angezeigt werden.

Obwohl Sie, wenn leere Methoden versteckt sind, diesen im Browser keinen Code hinzufügen können, können Sie dies trotzdem über das Kontextmenü des Code-Editors erreichen. Der „Wechseln auf“-Menüeintrag ist ein hierarchisches Untermenü, über das alle EventHandler, Methoden oder Eigenschaften des Fensters erreichbar sind.

Abb. 148: Selektieren einer leeren Methode, wenn leere Methoden versteckt sind



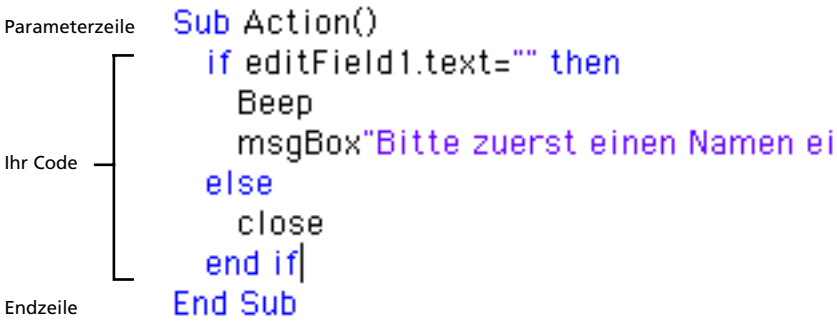
Mehr über Kontextmenüs finden Sie im Abschnitt „Kontext-Menüs“ auf Seite 195.

Anmerkung: Wenn ein neues Steuerelement angelegt wird, erhält es von REALbasic einen Namen. So erhält zum Beispiel der erste PushButton, den Sie einem Fenster hinzufügen, den Namen PushButton1. Ein solcher Name beschreibt den Objekttyp, aber nicht das Verhalten des Steuerelements. Sie sollten das Eigenschaftenfenster des Steuerelements dazu verwenden, Ihrem Steuerelement einen passenden Namen zu geben, der die Funktion des Steuerelements in der Anwendung wiederspiegelt. Der Browser zeigt neben jedem Steuerelement ein kleines Symbol an, um den Objekttyp oder die Klasse, von der das Steuerelement abgeleitet ist, zu verdeutlichen.

Methoden im Code-Editor

Event-Handler und Menu-Handler sind eigentlich auch Methoden, die speziell darauf ausgerichtet sind, bestimmte Aufgaben wahrzunehmen. Wird eine Methode im Code-Editor angezeigt, dann besteht sie aus drei Teilen: Der Parameterzeile, Ihrem Code und der Endzeile.

Abb. 149: Die drei Teile einer Methode

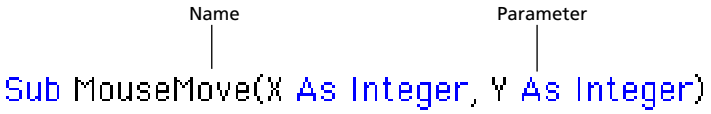


Die Parameterzeile

Die Parameterzeile zeigt Sub (Kurzform von Subroutine/Unterprogramm), wenn die Methode keine Werte zurückliefert, gefolgt vom Namen der Methode und dann folgen Klammern, in denen eventuelle Parameter stehen, die der Methode übergeben werden. In der Abbildung sehen Sie, wie das beim Event-Handler MouseMove aussieht. Dieser wird immer

dann aufgerufen, wenn die Maus über ein Objekt geschoben wird. Es werden zwei Koordinatenwerte als Parameter übergeben, damit man die Position der Maus feststellen kann.

Abb. 150: Die Teile der Parameterzeile



Mehr dazu siehe „Werte an Methoden übergeben“ auf Seite 166.

Liefert die Methode einen Wert zurück, dann heißt sie Function. Somit beginnt die Parameterzeile auch mit Function statt mit Sub. Sie benötigt außerdem noch eine Typangabe, die bestimmt, welchen Typ der Wert hat, den sie zurückliefert. In der Abbildung sieht man, wie das beim KeyDown Event-Handler eines EditField aussieht. Dieser wird immer dann aufgerufen, wenn der Benutzer in dem betreffenden EditField eine Taste drückt. Die Funktion liefert einen boole'schen Wert zurück. Ist dieser True, dann verhält sich REALbasic so, als ob die Taste nicht gedrückt worden wäre und das entsprechende Zeichen erscheint auch nicht im EditField. Damit lassen sich Zeichen für EditFields ausblenden, wenn beispielsweise nur Zahlen erlaubt sein sollen.

Abb. 151: Die Parameterzeile einer Funktion

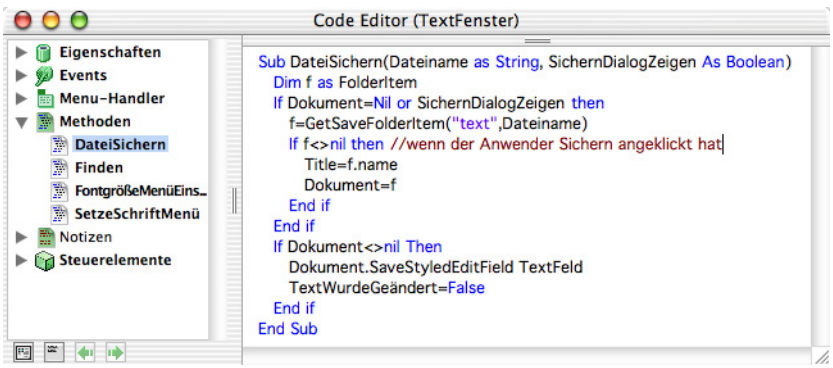
Mehr dazu siehe „Rückgabe von Werten aus Methoden“ auf Seite 167.

Eingabe des Programms im Code-Editor

Wenn Sie den Programmcode eingeben, hilft Ihnen REALbasic ein wenig. Zum einen rückt es Ihre If...Then, Select…Case Konstrukte und Schleifen beim Tippen für Sie ein, damit Sie einfacher sehen können, welche Codezeilen sich innerhalb eines bestimmten Ausdrucks befinden. Es verwendet ferner blauen Text für Schlüsselwörter und Datentypen. Folgende Abbildung zeigt ein Beispiel hierzu.

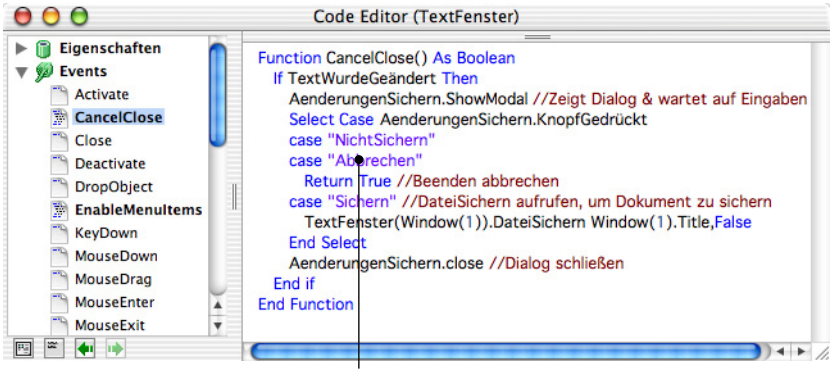
Außerdem erscheinen Kommentare, die Sie einfügen, in roter Schrift. Ein Textstring wird als nicht-ausführbarer Kommentar behandelt, wenn ihm ein Doppelslash (//), ein Apostroph (') oder das REM-Schlüsselwort vorangestellt wird. Kommentare müssen nicht unbedingt in einer eigenen Zeile stehen, wie folgende Abbildung zeigt.

Abb. 152: Ein Kommentar am Ende einer Zeile



Eine eigene Farbe wird auch für Textstrings verwendet, die Teil von ausführbarem Code sind. Dieser Text ist rosa, wie in folgender Abbildung gezeigt.

Abb. 153: Text in ausführbarem Code ist rosa



Text in ausführbarem Code hat eine andere Farbe als Kommentare

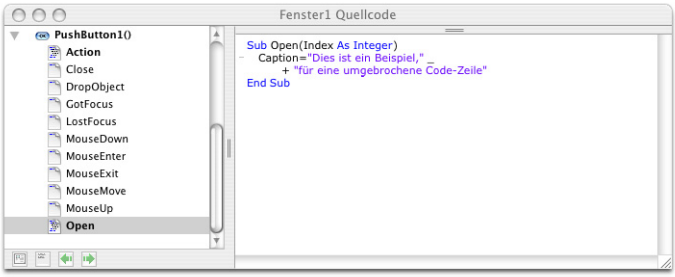
Eine Code-Zeile im Code-Editor umbrechen

Wenn Sie sehr lange Code-Zeilen eingeben, können Sie diese im Code-Editor umbrechen und auf zwei oder mehr Zeilen aufteilen. Beenden Sie eine solche Zeile mit einem Unterstrich und führen Sie die Zeile dann in der nächsten Zeile fort. Der REALbasic-Compiler wird nun erkennen, dass es sich bei der zweiten Zeile um eine fortgeführte Zeile handelt.

Wenn Sie zum Beispiel einer Eigenschaft einen sehr langen String zuweisen, können Sie diesen String auf zwei Zeilen aufteilen. Wenn Sie den Unterstrich verwenden, wird die folgende Zeile eingerückt, wie das folgende Beispiel zeigt. Der String, der hier zugewiesen wird, ist in zwei Zeilen aufgeteilt.

```
SaveChangesMessageDlg.Explanation="If you don't save now, you will"_
        +"lose all that important work you did since your last coffee break.
```

Abb. 154: Auf zwei Zeilen aufgeteilte Zeichenkette

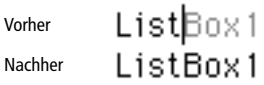


Beachten Sie, dass in diesem Beispiel der Unterstrich außerhalb der Anführungszeichen steht und somit nicht zum String gehört. Desweiteren wird der+ Operator verwendet, um die beiden Teilstrings miteinander zu verbinden. Dies ist notwendig, damit der Compiler nicht annimmt, dass der Unterstrich zu den String gehören soll. Wenn die Zeile, die Sie aufteilen wollen, keinen Text mit Anführungszeichen enthält, ist die Verwendung des + Operators nicht notwendig.

Auto-Vervollständigung

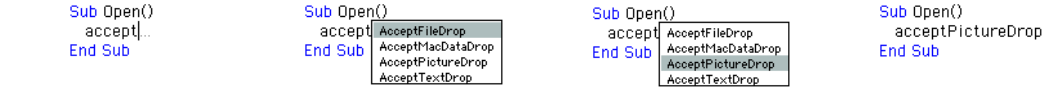
Während Sie tippen, versucht REALbasic zu erraten, was Sie tippen möchten und die Auto-Vervollständigung blendet in heller Schrift ein, was gemeint sein könnte. Wenn Sie die ersten Buchstaben eines REALbasic-Sprachenobjekts tippen – entweder ein eingebautes oder eine Variable, Methode oder Eigenschaft, die Sie erzeugt haben – zeigt REALbasic in heller Schrift seine Vermutung an. Ist die Vermutung richtig, drücken Sie die Tab-Taste, um den Eintrag zu vervollständigen. Liegt REALbasic falsch, tippen Sie einfach weiter.

Abb. 155: Auto-Vervollständigung



Wenn REALbasic mehrere passende Objekte findet, zeigt es stattdessen Punkte (...) an. Wenn Sie die Tab-Taste drücken, erscheint ein Kontextmenü mit den Vorschlägen und Sie können den passenden auswählen. Dies zeigt folgende Abbildung.

Abb. 156: Auswahl bei Auto-Vervollständigung



Der Anwender tippt „accept". Es erscheinen drei Punkte.

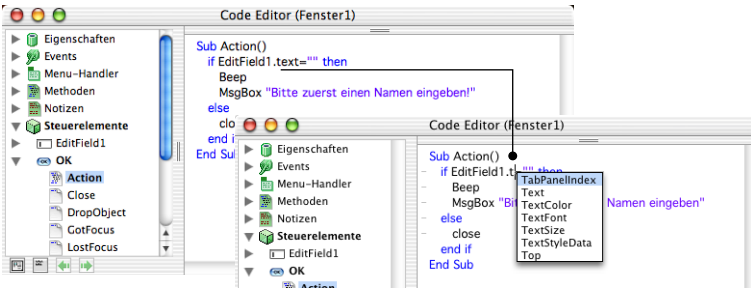
Er drückt die Tab-Taste, um das Kontextmenü aufzurufen.

Er wählt mit Hilfe der Cursortasten (auf/ab) den gewünschten Eintrag aus und drückt danach Return.

REALbasic vervollständigt die Eingabe.

Auto-Vervollständigung funktioniert auch inmitten eines Ausdrucks. Ganz gleich, wo sich der Eingabecursor im Ausdruck befindet, können Sie immer die Tab-Taste drücken, um sich ein Popup-Menü mit möglichen Begriffen anzeigen zu lassen. In der folgenden Abbildung befindet sich der Eingabecursor nach dem „t" in Text.

Abb. 157: Auto-Vervollständigung innerhalb eines Ausdrucks



Auto-Vervollständigung funktioniert auch bei selbstdefinierten Eigenschaften, Methoden, Funktionen und Events.

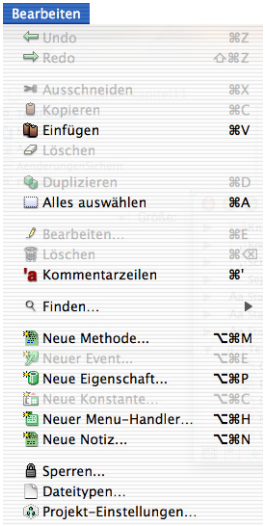
Im folgenden Beispiel erzeugt der Anwender eine neue Eigenschaft für ein Fenster. REALbasic schlägt korrekt vor, dass es sich beim Objekttyp um ein FolderItem handelt.

Abb. 158: Auto-Vervollständigung im Dialogfenster Neue Eigenschaft

Das Bearbeiten-Menü

Während Sie Code eingeben, sind die Standard-Befehle des Bearbeiten-Menüs wie Ausschneiden, Kopieren und Einfügen verfügbar. Der Code-Editor unterstützt auch Undo und Redo (Shift-⌘-Z oder Shift-Strg-Z). Die Funktion „Kommentarzeilen" (⌘-' oder Strg-') ist ebenfalls sehr nützlich. Wenn sie auf Quelltextzeilen angewendet wird, werden diese auskommentiert, wenn sie auf Kommentarzeilen angewendet wird, werden diese in Code zurück gewandelt. Folgende Abbildung zeigt das Bearbeiten-Menü während der Code-Eingabe.

Abb. 159: Das Bearbeiten-Menü



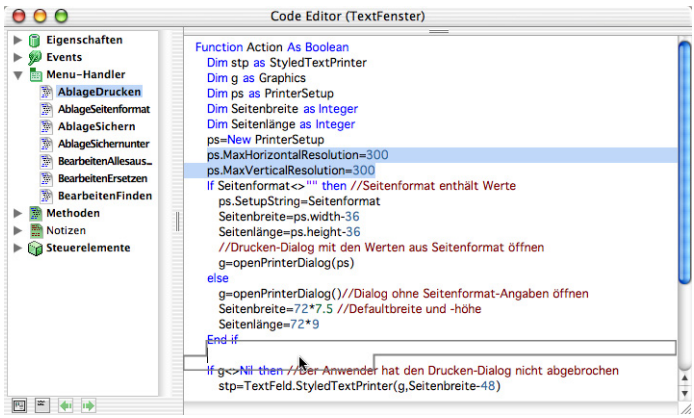
Hinweis: Unter Mac OS X befindet sich das Einstellungen-Menu von REALbasic im REALbasic-Menu.

Drag & Drop

Der Code-Editor bietet volle Drag & Drop-Unterstützung. Innerhalb einer Methode können Sie:

- Text bewegen: Markieren Sie den Text, der bewegt werden soll und draggen Sie ihn an die gewünschte Position, die durch eine Einfügemarke angezeigt wird.
- Text kopieren: Markieren Sie den Text, der kopiert werden soll, halten Sie die alt-Taste gedrückt und draggen Sie ihn an die gewünschte Position.

Abb. 160: Codezeilen per Drag&Drop bewegen



Sie können markierten Code auch in ein anderes Programm, das Drag&Drop unterstützt, draggen oder von dort per Drag&Drop holen.

Sie können auch einen Text-Clip vom Schreibtisch in den Code-Editor draggen oder selektierten Text auf den Schreibtisch draggen, um einen Text-Clip zu erzeugen.

Die Online-Referenz (⌘-1 oder F1 unter Windows) enthält zahlreiche Beispiele mit REALbasic-Code. Jedes Beispiel ist von einem gepunkteten Rechteck umgeben. Jedes Code-Rechteck kann in den Code-Editor gedraggt werden. Sie können jedoch keine einzelnen Zeilen innerhalb des Rechtecks für das Drag & Drop selektieren. Folgende Abbildung zeigt ein Beispiel für einen Bereich, den man in das Code-Editor ziehen kann.

Abb. 161: Ein Code-Beispiel in der Online-Referenz

Zwischen Code-Editoren hin- und herdraggen

Wenn Sie zwei oder mehr Code-Editoren geöffnet haben (zum Beispiel von einem Fenster und einem Modul), können Sie Eigenschaften oder Methoden von einem Code-Editor zum anderen draggen, indem Sie die alt-Taste gedrückt halten und eine Eigenschaft oder Methode vom einem Fenster ins andere ziehen. Events werden als Methoden kopiert. Sie können ebenso Code aus dem Code-Editor in den Browser-Bereich draggen. Indem Sie ein Code-Schnipsel in die Liste draggen, wird eine Methode ohne Rückgabewerte oder Parameter hinzugefügt. Für den Fall, dass Sie eine einzelne Variable deklarieren, wird stattdessen eine Eigenschaft hinzugefügt.

Code über Schnellkopie einfügen

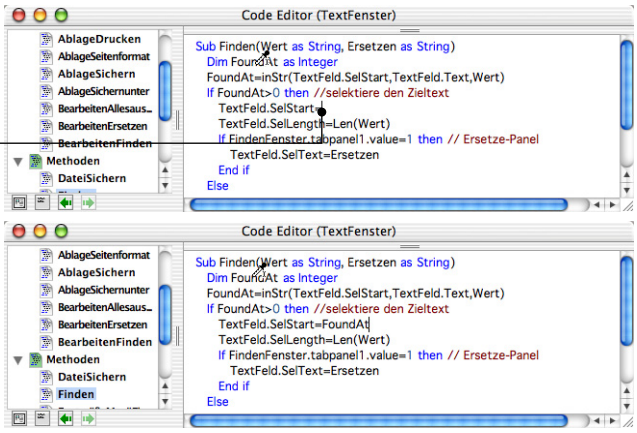
Der Code-Editor bietet eine Möglichkeit, Text sehr einfach und schnell zu kopieren. Wenn Sie die Shift- und Alt-Taste drücken, ändert sich der Mauszeiger in eine Pipette . An der Stelle, an der sich der Schreibcursor befindet, können Sie nun das Wort einfügen, auf das Sie mit der Pipette klicken. Dies zeigt folgende Abbildung.

Abb. 162: Code mit der Schnellkopie einfügen

Platzieren Sie die Einfügemarke an der gewünschten Position. Klicken Sie mit gedrückter alt-Taste auf Ausdruck, den Sie einfügen möchten.

Einfügemarke

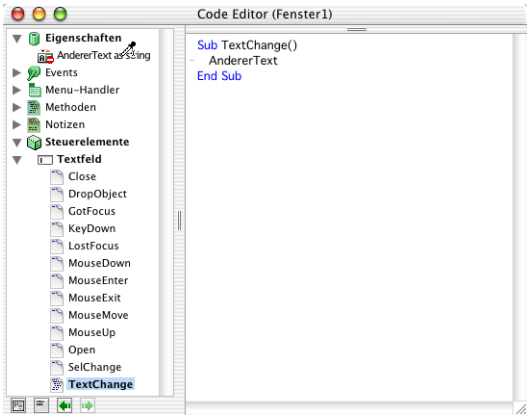
Nach dem alt-Klick erscheint der Ausdruck an der Einfügemarke.



Einfügen eines Eigenschafts- oder Methodennamens mittels Schnellkopie

Die Auto-Vervollständigen-Funktion macht es einfach, im Code-Editor Eigenschaften- oder Methodennamen einzugeben. Es geht mit der Schnellkopie aber noch einfacher: Klicken Sie einfach bei gleichzeitig gedrückter Shift- und alt-Taste einen Eintrag der Objektliste an, schon wird er an der Einfügemarke eingefügt.

Abb. 163: Einfügen des Eigenschaftsnamens AndererText mittels Schnellkopie

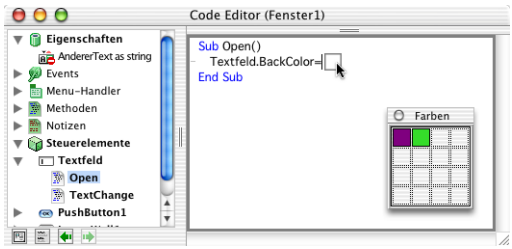


Drag & Drop von Farben

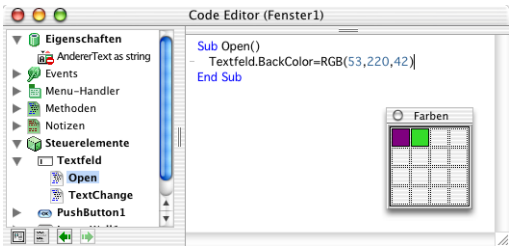
Sie können Objekten, die Farbeigenschaften besitzen, Farben per Drag & Drop aus dem Farbenfenster in den Code-Editor zuweisen. Dazu sollten Sie in den **Einstellungen** für den Code-Editor **Blende automatisch aus: Farben** deaktivieren. Ansonsten versteckt REALbasic automatisch das Farbenfenster, wenn Sie den Code-Editor verwenden.

Wenn Sie eine Farbe draggen, enthält das gedraggte Objekt eine Text-Repräsentation der Farbe in Form eines RGB-Wertes. Sie müssen lediglich Ihre Farbauswahl aus dem Farbenfenster an die passende Stelle im Code-Editor zu draggen. Am &c-Operator ist zu erkennen, dass es sich um eine Farbe handelt.

Abb. 164: Eine Farbe per Drag & Drop zuweisen



Der BackColor-Eigenschaft wird eine Farbe zugewiesen, indem eine Farbe vom Farbenfenster auf den Code-Editor gedraggt wird.



Beim Loslassen der Maustaste an der Einfügemarke erscheint der dem Farbwert entsprechende Code.

Drag & Drop von Steuerelementen

Sie können dem Code-Editor den Namen jedes beliebigen Steuerelements durch Drag & Drop aus der Steuerelementepalette hinzufügen. Dazu müssen Sie in den **Einstellungen** für den Code-Editor **Blende automatisch aus:Steuerelemente** deaktivieren. Ansonsten versteckt REALbasic die Steuerelementepalette wenn Sie den Code-Editor verwenden.

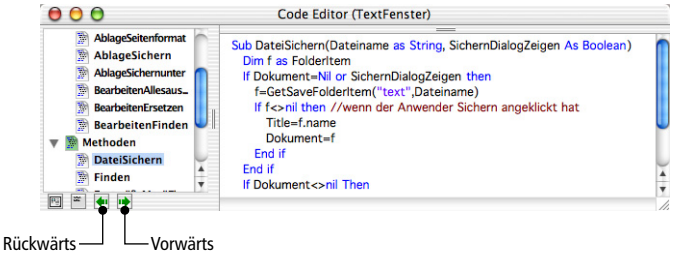
Um dem Code-Editor den Namen eines Steuerelements hinzuzufügen, draggen Sie einfach ein Steuerelement aus der Steuerelementepalette in den Code-Editor, genauso als ob Sie einem Fenster eine Instanz eines Steuerelements hinzufügen wollten. Wenn Sie in diesem Stil vorgehen, fügt REALbasic den Namen des gedraggten Steuerelements im Code ein, aber nicht den Namen einer Instanz dieses Steuerelement-Typs. Wenn Sie also einen PushButton in den Code-Editor draggen, dann fügt REALbasic den Text **PushButton** neben der Einfügemarke ein. Wenn Sie sich auf eine Instanz dieses Steuerelements beziehen möchten (vorausgesetzt Sie haben beispielsweise PushButtons mit der Bezeichnung **PushButton1**, **PushButton2** etc.), müssen Sie eine entsprechende Änderung im Text vornehmen.

Sie können diese Programmfunktion dazu einsetzen, Instanzen von Steuerelementen per Code zu erzeugen. Das Serial- und das TCPSocket-Steuerelement sind beispielsweise in einem Fenster nicht sichtbar. Stattdessen können Sie diese Steuerelemente über den Open-Event-Handler des Fensters erzeugen. Wenn Sie möchten, kann Ihnen die Technik des Drag & Drop dabei behilflich sein, den Code zu schreiben, der eine solches Steuerelement instanziiert.

Die Navigationsknöpfe

Der Code-Editor erlaubt es, von Methode zu Methode zu springen, indem man auf die Objekte und Events in der Objektliste klickt. REALbasic merkt sich dabei die Reihenfolge, in der die Methoden angezeigt wurden und macht es dadurch möglich, diese Schritte mit den Vorwärts- und Rückwärts-Navigationsknöpfen zurückzuverfolgen.

Abb. 165: Die Navigationsknöpfe des Code-Editors

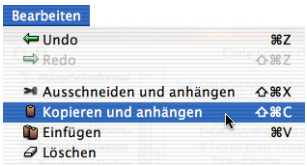


Diese Knöpfe arbeiten genau so wie die Knöpfe in einem Web-Browser. Der Rückwärts-Knopf ist wählbar, wenn Sie zu einer zweiten Methode gehen, und der Vorwärts-Knopf ist wählbar, nachdem Sie das erste mal den Rückwärts-Knopf benutzt haben.

Ausschneiden/Kopieren und Anhängen

Wenn Sie im Code-Editor arbeiten, gibt es eine nützliche Erweiterung zum normalen Ausschneiden und Kopieren. Wenn Sie zusätzlich die Shift-Taste drücken, werden diese Befehle zu „Ausschneiden und anhängen“ und „Kopieren und anhängen“. Beide Befehle hängen den selektierten Text an den Text in der Zwischenablage an, statt den dort bereits vorhandenen zu ersetzen.

Abb. 166: Ausschneiden/Kopieren und Anhängen

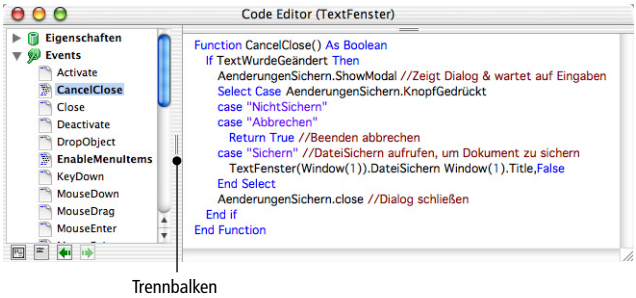


Durch diese Erweiterung können Sie mehrere, nicht zusammenhängende Textteile sammeln und als Ganzes an der gewünschten Stelle auf einmal einfügen. Die Tastaturbefehle sind wie in der Abbildung ersichtlich ⌘-Shift-X bzw. ⌘-Shift-C (Windows: Strg-Shift-X bzw. Strg-Shift-C).

Mehr Platz im Code-Editor

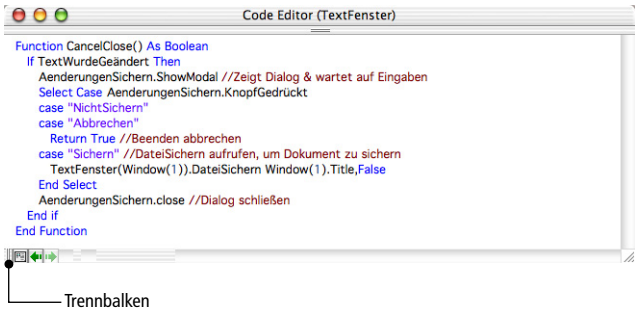
Manchmal kann es sein, dass Sie mehr Platz benötigen, aber das Fenster nicht vergrößern können. Sie können die Schrift im Code-Editor verkleinern, indem Sie den Menüpunkt **Bearbeiten/Einstellungen** (unter Mac OS X **REALbasic/ Einstellungen**) aufrufen. Außerdem können Sie den Trennbalken zwischen Editor und Browser nach links verschieben.

Abb. 167: Trennbalken des Code-Editors



Wenn Sie den Trennbalken ganz nach links schieben, dann wird der Browser hinter einem kleinen Quadrat links unten im Editorfenster versteckt, das sich einfach wieder nach rechts ziehen lässt, um den Browser wieder einzublenden.

Abb. 168: Code Editor mit verstecktem Browser



Ist der Browser einmal zusammengeklappt, kann man ihn mit Shift-Tab wieder ausklappen und mit erneutem Shift-Tab verstecken. Dabei springt der Cursor auch gleich auf den Browser, wodurch man mit den Cursor-Tasten Objekte im Browser ansteuern kann. Shift-Tab reagiert nur so, nachdem das Browser-Fenster einmal auf diese Weise verkleinert wurde. Sonst wird Shift-Tab dazu verwendet, den Cursor vom Editorfenster in das Browser-Fenster zu setzen. Dies ist so, weil ein Benutzer, der die Maus dazu verwendet, im Browser Objekte zu selektieren, trotzdem den Fokus auf dem Editor haben will, so dass die nächsten Tastatureingaben weiterhin in Editorfenster gehen und nicht in den Browser.

Anmerkung: Setzt man den Cursor auf eine Zeile, die mit einem Dreieck markiert ist, dann lässt sich das Dreieck mit der Befehlstaste (Windows: Strg-Taste) und den Cursortasten nach rechts oder links ein- und ausklappen.

Mehrere Event-Handler gleichzeitig anzeigen

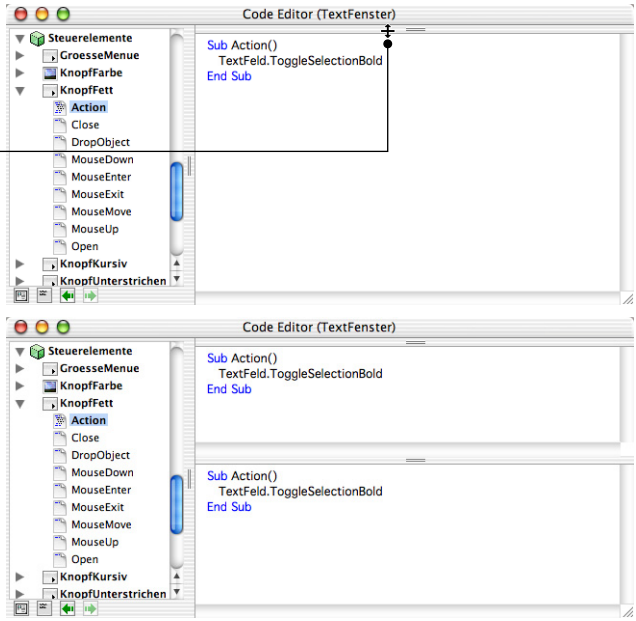
Gemäß Voreinstellung wird im editierbaren Bereich des Code-Editors nur der Code für einen Event-Handler oder eine Methode angezeigt. Dabei handelt es sich um den Event-Handler, den Sie im Browserbereich des Code-Editors angewählt haben. Wenn Sie sich mehr als einen Event-Handler oder eine Methode anzeigen lassen möchten, dann können Sie den editierbaren Bereich des Code-Editors in zwei oder mehr Bereiche unterteilen, indem Sie die Trennleiste draggen. Die Trennleiste befindet sich direkt unter der Titelleiste und ist durch zwei horizontale Linien gekennzeichnet. Bewegen Sie den Mauszeiger über die Trennleiste, bis er sich in ein Symbol zum Verschieben der Trennleiste verwandelt. Draggen Sie anschließend die Trennleiste nach unten, um einen neuen abgetrennten Editierbereich anzulegen. Sie kön-

nen auf diese Art und Weise mehrere Editierbereiche erzeugen. Weitere Editierbereiche können auch erzeugt werden, indem Sie auf die Trennleiste unter der Titelleiste doppelklicken. Wenn Sie auf die untere Trennleiste doppelklicken, wie in der unteren Abbildung gezeigt, schließen Sie den unteren Editierbereich.

Abb. 169: Erzeugen eines zweiten Eingabebereiches im Code-Editor

Selektieren Sie die Trennleiste und draggen Sie sie nach unten.

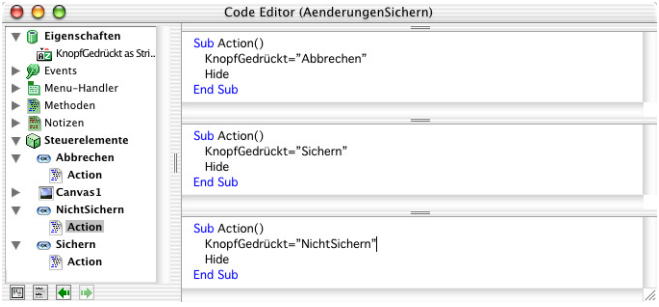
REALbasic erzeugt einen weiteren Eingabebereich und zeigt in diesem ebenfalls den oben aktiven Event-Handler an. Um einen anderen Event-Handler bearbeiten zu können, setzen Sie die Eingabemarke in den unteren Eingabebereich und wählen dann im Browser einen anderen Event-Handler aus.



Indem Sie diesen Vorgang wiederholen, können Sie weitere Unterteilungen vornehmen. Um den Event-Handler für den Editierbereich auszuwählen, setzen Sie die Einfügemarke in den entsprechenden Editierbereich und wählen den Event-Handler im Browserbereich aus.

Sie können die Größe der einzelnen Editierbereiche variieren, indem Sie die Trennleiste zwischen zwei vorhandenen Editierbereichen nach oben oder unten verschieben.

Abb. 170: Drei Eingabebereiche im Code-Editor



Der aktive Editierbereich ist derjenige mit der Einfügemarke. Im obigen Beispiel befindet sich die Einfügemarke im oberen Editierbereich und zeigt den Action-Event für den **KnopfFett** an. Bei einem Klick des Benutzers auf einen anderen Event im Browser würde der Code dieses Events statt des **KnopfFett**-Codes angezeigt werden. Beachten Sie dies, damit Sie während der Arbeit mit mehreren Editierbereichen nicht durcheinander kommen.

Zwischen den Editierbereichen können Sie die Techniken Drag & Drop und Schnellkopie anwenden.

Ein Fenster aus dem Code-Editor heraus öffnen

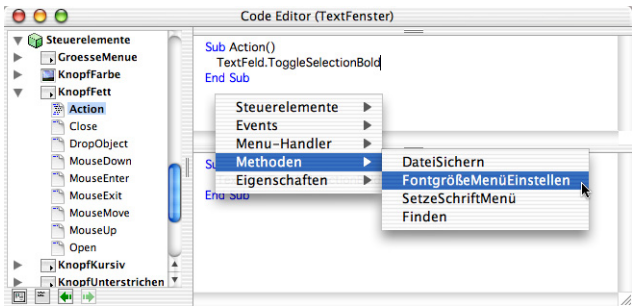
Wenn Sie im Code-Editor arbeiten, der zu einem Fenster gehört, können Sie das Fenster direkt aus dem Code-Editor heraus öffnen. Das erste Icon in der horizontalen Scroll-Leiste  öffnet das Fenster. Wenn Sie im Code-Editor arbeiten, der nicht zu einem Fenster gehört (z.B. eine Klasse, die Sie Ihrem Projekt über „Ablage/Neue Klasse“ hinzugefügt haben), ist das Icon nicht anwählbar.

Kontext-Menüs

Die Methoden im Browser können vom Code-Editor aus auch über Kontext-Menüs erreicht werden. Kontext-Menüs sind – wie der Name andeutet – automatisch auf den Kontext orientiert, in dem sie aufgerufen werden. Wenn Sie bei gedrückter ctrl-Taste (Windows: rechte Maustaste) auf die Arbeitsfläche des Editorfensters klicken, werden die verwendeten Komponenten aus dem Browser-Fenster in einem Menü eingeblendet. Das ist besonders dann geschickt, wenn der Browser oder die leeren Methoden gerade ausgeblendet sind.

Das Kontextmenü des Code-Editors ist ein hierarchisches Menü mit zwei Einträgen auf der obersten Hierarchieebene: **Neu** und **Wechseln Auf**. Das **Neu**-Menü hat ein Untermenü, über das Sie im Code-Editor ein neues Element anlegen können. Bei Code-Editoren, die zu einem Fenster gehören, können Sie eine neue Methode, Eigenschaft, einen neuen Menü-Handler oder eine Notiz hinzufügen. Bei Code-Editoren, die zu einer Klasse gehören, können Sie zusätzlich ein neues Event ins Leben rufen.

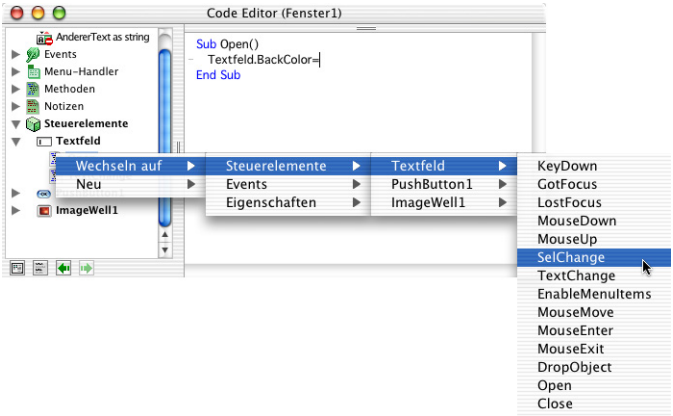
Abb. 171: Kontext-Menüs im Code Editor



Diese Menübefehle stehen auch im **Bearbeiten**-Menü der IDE zur Verfügung.

Über das **Wechseln Auf**-Menü können Sie im Editierbereich direkt die Event-Handler von Steuerelementen, Menu-Handlern, Fenster-Events, Fenster-Methoden und Fenster-Eigenschaften einblenden. Dabei listet das **Wechseln Auf**-Untermenü immer alle Handler auf, die einem Steuerelement oder Fenster zur Verfügung stehen – nicht nur diejenigen, für die bereits Code existiert.

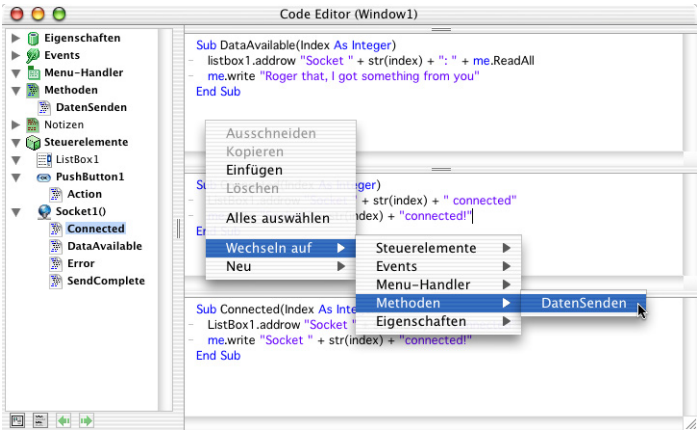
Abb. 172: Das „Wechseln Auf"-Kontextmenü des Code-Editors



Wenn Sie den Code-Editor in mehrere Arbeitsflächen aufgeteilt haben, können Sie den Inhalt einer ausgewählten Arbeitsfläche ändern, indem Sie den Textcursor auf die Arbeitsfläche setzen, die verändert werden soll, und sich dann das Kontextmenü anzeigen lassen.

Wenn sich der Mauszeiger im Code-Editor befindet, während Sie das Kontextmenü aufrufen, dann erscheinen oberhalb der **Neu** und **Wechseln Auf**-Menüeinträge auch die Standard-Befehle des **Bearbeiten**-Menüs. Wenn Sie das Kontextmenü innerhalb des Code-Editors über einem Text aufrufen, erscheint über den Einträgen des Bearbeiten-Menüs auch den Menüeintrag „Suchen nach". Mit diesem können Sie nach dem markierten Text oder dem Text unter dem Mauszeiger suchen. Für weitere Informationen lesen Sie auch den Abschnitt „Objekt- und Variablen-Deklaration suchen" auf Seite 197.

Abb. 173: Das Kontextmenü des Code-Editors in der Arbeitsfläche

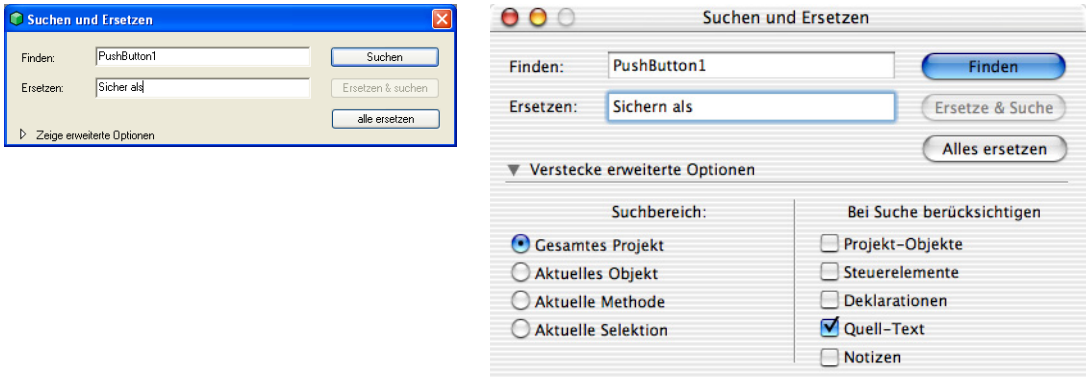


Suchen und Ersetzen

Mit dem Suchen/Ersetzen-Dialog (**Bearbeiten/Suchen/Suchen** oder ⌘-F (Mac) bzw. Strg-F (Windows)) können Sie bestimmte Texte in Ihrem Code suchen und durch andere ersetzen. Wenn Sie im aktuellen Code suchen, dann beginnt die Suche ab der Einfügemarke. Wenn die Suche das Ende des Codes erreicht hat, wird sie am Anfang des Codes bis zum Erreichen der ursprünglichen Einfügemarke fortgesetzt.

REALbasic erlaubt die Suche im aktuellen Modul, auch wenn das Code-Editorfenster nicht geöffnet ist. In diesem Falle beginnt die Suchroutine in dem im Projektfenster ausgewählten Objekt mit der Suche.

Abb. 174: Der Finden/Ersetzen-Dialog ohne und mit erweiterten Optionen



Sie haben die Auswahl aus folgenden Suchkriterien:

Projekt-Objekte: Sucht in den Namen der Projekt-Objekte im Projektfenster. Die Suche beschränkt sich dabei nicht auf Klassen und Fenster, sondern beeinhaltet alle darin enthaltenen Elemente inklusive Ordner, Resourcen, Bilder, etc.

Steuerelemente: Durchsucht die Namen der Steuerelemente und zeigt sie im Code-Editor an.

Deklarationen: Durchsucht Menu-Handler, deklarierte Eigenschaften des Benutzers und Methoden- Deklarationen inklusive der Parameter und zeigt sie im Code-Editor-Browser an.

Quelltext: Durchsucht Ihren Code, so wie Sie ihn im Code-Editor eingegeben haben. Sie können den Suchbereich einschränken, indem Sie einen anderen Option als „Gesamtes Projekt“ unter „Suchbereich“ auswählen.

Notizen: Durchsucht die Notizen.

Auch der Suchbereich lässt sich hier festlegen.

Tabelle 5. Der Suchbereich

Gesamtes Projekt	Suchen und Ersetzen durchsucht das gesamte Projekt.
Aktuelles Objekt	Suchen und Ersetzen durchsucht die Methoden des aktuellen Fensters, Moduls oder der aktuellen Klasse
Aktuelle Methode	Suchen und Ersetzen betrifft nur die aktuell angezeigte Methode
Aktuelle Selektion	Suchen und Ersetzen betrifft nur den aktuell ausgewählten Code

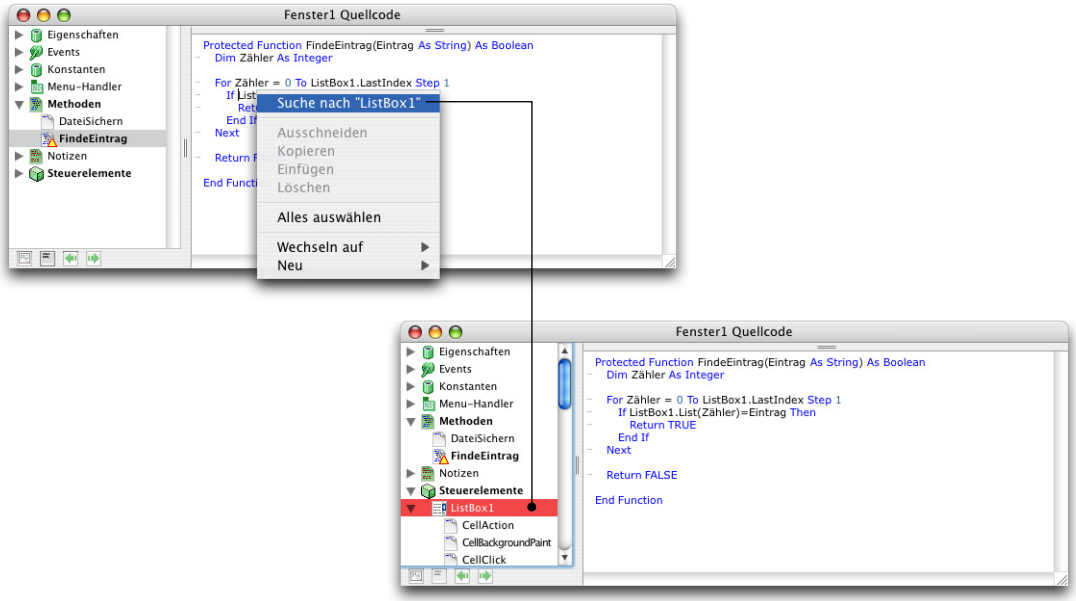
Mit den Knöpfen in diesem Dialog können Sie nach dem nächsten Auftreten des Textes suchen, den im Code-Editor markierten Text durch den im Feld „Ersetzen“ ersetzen lassen oder alle Vorkommen im eingestellten Suchbereich ersetzen lassen. Die Tastaturbefehle sind ⌘-F (Windows: Strg-F) für „Suchen“, ⌘-G (Strg-G) für „Erneut suchen“ und ⌘-L (Strg-L) für „Ersetzen und Suchen“.

Objekt- und Variablen-Deklaration suchen

Wenn Sie nach der Deklaration eines Objekts oder einer Variable suchen wollen, können Sie dies bequem über den Suche-Befehl im Kontextmenü des Code-Editors. Der „Suche nach…“-Menüeintrag findet zum Beispiel eine Dim-Deklaration, eine Eigenschaft, eine Konstante oder eine Methoden-Deklaration. Dabei werden auch Code-Teile anderer Objekte im Projekt durchsucht, weshalb dieser Befehl sich bestens dafür eignet, Deklarationen zu suchen, die in einem anderen Objekt enthalten sind.

Um das Kontextmenü zu benutzen, platzieren Sie den Cursor in dem gesuchten Ausdruck und führen einen Control-Klick (Rechtsklick unter Windows) aus und wählen den Menüeintrag „Suchen nach…“. Befindet sich die Deklaration in einem anderen Objekt, wird ein neues Code-Editor Fenster geöffnet, um die gesuchte Deklaration anzuzeigen.

Im folgenden Beispiel wird die Deklaration des Ausdrucks „ListBox1“ gesucht. Es stellt sich heraus, dass es sich dabei um ein Steuerelement handelt.



REALbasic durchsucht zuerst den geöffneten Code-Editor. Falls es nicht fündig wird, werden die anderen Teile des Projekts durchsucht. Gibt es mehrere Deklarationen des gesuchten Ausdrucks, können diese mit Hilfe des Menüeintrags **Bearbeiten/Suchen/Weitersuchen** nacheinander gefunden werden.

Drucken des Programmcodes

Um den Quelltext Ihres Programms auszudrucken, wählen Sie **Datei/Drucken** (⌘-P oder Strg-P). In der folgenden Dialogbox können Sie näher bestimmen, was ausgedruckt werden soll.

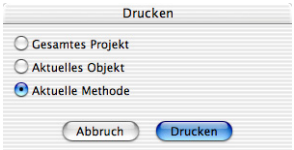
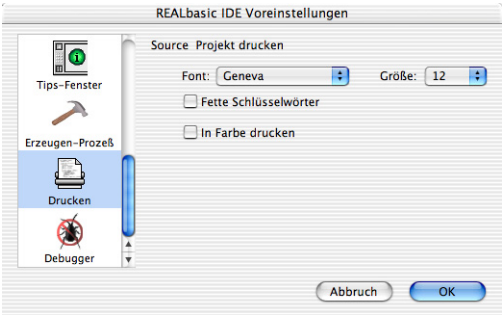


Tabelle 6. Optionen der Druckdialogbox

Option	Beschreibung
Gesamtes Projekt	Ausdruck des gesamten Projektquelltexts.
Aktuelles Objekt	Ausdruck des Codes des gerade angezeigten Fensters, Objekts oder Moduls.
Aktuelle Methode	Ausdruck der gerade angezeigten Methode.

Über **Bearbeiten/Einstellungen** (oder **REALbasic/REALbasic Einstellungen** unter Mac OS X) können Sie die Schriftart und -größe für das Drucken einstellen. Außerdem legen Sie hier fest, ob Schlüsselwörter fett gedruckt werden sollen und ob die für die Bildschirmdarstellung eingestellten Farben auch für den Ausdruck verwendet werden sollen.

Abb. 175: Voreinstellungen für das Drucken



Export/Import von Programmcode

Es ist einfach, von Ihnen angelegte Objekte zu exportieren oder zu importieren. Leicht können Quelltextteile, Fenster, Menüs, Sounds, Bilder, QuickTime-Filme, REAL-Datenbanken und Resource-Dateien in Ihr Projekt importiert werden.

Externe Projektelemente

Wenn Sie ein Objekt in mehreren Projekten verwenden möchten, können Sie es als externes Projekt-Objekt importieren. Das Objekt verbleibt auf der Festplatte. Änderungen, die von einem anderen Projekt an ihm vorgenommen werden, werden automatisch in das aktuelle Objekt übernommen, wenn Sie es erneut öffnen. Sie können jedoch nicht mehr als ein Projekt gleichzeitig öffnen, das auf das gleiche externe Objekt zugreift. Um einem Projekt ein externes Element hinzuzufügen, halten Sie die Befehls- und alt-Taste gedrückt (Ctrl-Shift unter Windows) während Sie das Element vom Schreibtisch auf das Projektfenster ziehen. Der Name des externen Objekts wird im Projektfenster kursiv dargestellt.

Importieren

Um eine Datei in Ihr Projekt zu importieren, ziehen Sie diese einfach vom Desktop auf Ihr Projektfenster und lassen sie dort fallen. Alternativ können Sie den Menüpunkt **Ablage/Import** verwenden und die Datei in der Dateiauswahlbox auswählen.

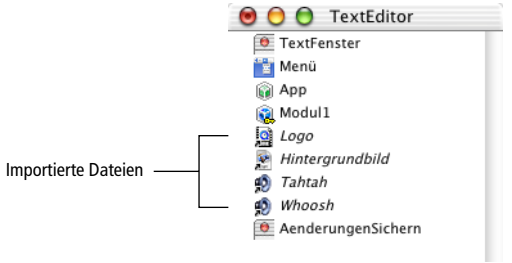
Wenn es sich um Code handelt, öffnen Sie die Methode, in die Sie den Code importieren möchten und ziehen Sie einen Text-Clip in die Methode hinein. Sie können nicht den Menüpunkt **Ablage/Import** verwenden, um Textdateien in eine Methode zu importieren.

Eine Datei, die Sie dem Projektfenster hinzugefügt haben, löschen Sie, indem Sie sie selektieren und die Backspace-Taste drücken oder „Bearbeiten/Löschen“ aufrufen. Sie können ein Objekt im Projektfenster auch löschen, indem Sie einen ctrl-Klick (Windows: Rechtsklick) darauf ausführen und aus dem Kontextmenü „Löschen“ auswählen.

Hinweis: Wenn Sie ein REALbasic-Objekt durch Ziehen in das Projektfenster importieren, ersetzt es automatisch ohne Nachfrage ein Objekt gleichen Namens.

Manche Objekte werden in Ihr Projekt kopiert. Andere verbleiben auf der Platte und es wird nur ein Alias in das Projekt gesetzt, der auf diese Datei zeigt. Im Projektfenster wird ein Alias kursiv dargestellt. Im folgenden Beispiel wurden ein QuickTime-Film (Logo), ein PICT-Bild (Hintergrundbild) und mehrere Sound-Dateien in das Projekt importiert.

Abb. 176: Ein Projektfenster mit Aliasen von Dateien



Wenn Sie aus Ihrem Projekt ein eigenständiges Programm erzeugen, werden die meisten dieser Dateien in dieses Programm kopiert. Folgende Tabelle zeigt die verschiedenen benutzbaren Dateitypen.

Tabelle 7. Wie REALbasic Dateien behandelt

Dateityp	Kopie im Projekt?	Kopie in eigenständigem Programm?
Bitmap, PICT, JPEG, GIF	Nein	Ja
Cursors	Ja	Ja
PowerPC Shared Libraries	Nein	Nein
QuickTime Movies	Nein	Ja
*REAL Datenbanken**	Nein	Nein
REALbasic Classes	Ja	Ja
REALbasic Menubars	Ja	Ja
REALbasic Module	Ja	Ja
REALbasic Windows	Ja	Ja
Resources	Nein	Ja
Sounds	Nein	Ja
AppleScripts	Nein	Ja

*Datenquellen wie z.B. REAL-Datenbanken erscheinen in normaler Schrift im Projektfenster, obwohl die Daten außerhalb gespeichert sind. Die Verbindungsinformation zur Datenquelle wird im Projekt gespeichert.

Weil REALbasic in Fällen, in denen das Original einer Datei auf der Platte bleibt, einen Alias verwendet, kann die externe Datei umbenannt und sogar verschoben werden. Bestenfalls beim Verschieben des Projekts und/oder der Referenzdateien auf eine andere Platte kann es vorkommen, dass REALbasic die Dateien nicht mehr findet. In solchen Fällen fragt REALbasic nach dem Speicherort der vermissten Dateien.

Wenn Sie ein Bild in Ihr Projekt importieren wird es in den Speicher geladen, sobald Sie das Programm starten, egal ob das Bild verwendet wird oder nicht.

Wenn Sie viele große Bilder verwenden, kann es passieren, dass Ihr Programm sich seltsam verhält oder ihm der Speicher ausgeht. Wenn Sie große Bilder verwenden, sollten Sie diese extern lassen und mit der GetFolderItem-Methode laden oder sie in einer PICT-Resource speichern und diese über die GetPicture-Methode von ResourceFork laden. Denken Sie daran, dass das Speichern in einer PICT-Resource nicht plattformübergreifend funktioniert, so dass dies bei einem Programm für Windows nicht möglich ist.

Alle Dateitypen außer PowerPC-Shared Libraries und REAL Datenbanken werden in das ausführbare Programm compiliert. Sie müssen diese also nicht beilegen, wenn Sie Ihr Programm weitergeben.

Exportieren

Der Code für Methoden, Events, Konstanten, Eigenschaften etc. kann per Drag & Drop aus dem Code-Editor gezogen und als Textclip abgelegt werden. Zu diesem Zweck können Sie entweder Codezeilen im Editor selektieren oder den Namen eines Objekts im Browser auswählen und dann draggen. Wenn Sie den Namen eines Objekts draggen, wird im Textclip der gesamte zum Objekt gehörende Code abgelegt.

Außerdem können Sie Quelltext in einer Textdatei ablegen oder im REALbasic-Format. Welches Format Sie wählen, hängt von der späteren Verwendung der Datei ab. Soll der Quelltext später in einem Text (z.B. zur Dokumentation) verwendet werden, dann exportieren Sie ihn im Textformat über „Datei/Quelltext exportieren“ oder draggen Sie ihn direkt in das Programm, mit dem Sie die Dokumentation schreiben.

Soll ein Fenster, ein Modul, eine Klasse oder eine Menüleiste zur späteren Verwendung in einem anderen REALbasic-Projekt exportiert werden, können Sie das Objekt einfach aus dem Projektfenster auf den Desktop ziehen. Auf dem Desktop haben exportierte Objekte ihre eigenen REALbasic-Icons.

Abb. 177: Icons exportierter REALbasic-Objekte

Sie können REALbasic-Objekte aber auch über den Export-Befehl exportieren. Öffnen Sie hierzu das entsprechende Objekt oder selektieren Sie es im Projektfenster. Wählen Sie dann entweder im Ablage- oder im Kontextmenü des Objekts den Eintrag **Fenster/Menü/Modul/Klasse exportieren**. In der Dialogbox klicken Sie den **Sichern**-Knopf.

Verschlüsseln des Quelltextes

Wenn Sie ein exportiertes Objekt an andere weitergegeben wollen, Ihr Quelltext aber nicht lesbar und bearbeitbar sein soll, rufen Sie den Menüpunkt **Bearbeiten/Sperren** auf, um Ihr Objekt zu schützen, bevor Sie es exportieren. Das Icon eines gesperrtes Objekts hat ein kleines Schlüssel-Symbol in der rechten unteren Ecke. Wenn ein gesperrtes Objekt exportiert wird, unterscheidet sich sein Icon nicht von einem ungesperrten. Sie können ein Fenster sperren und entsperren, so lange es sich in Ihrem Projekt befindet. Ein verschlüsseltes Fenster kann nicht im Fenstereditor angezeigt werden und niemand hat Zugriff auf Code, der zu diesem Fenster oder einem seiner Steuerelemente gehört. Eine genaue Beschreibung, wie man Objekte schützt, finden Sie im Abschnitt „Fenster verschlüsseln“ auf Seite 96.

Beim Verschlüsseln eines Objekts geben Sie ein Passwort an, mit dem Sie das Objekt später wieder entschlüsseln können.

Um ein Objekt zu verschlüsseln, gehen Sie folgendermaßen vor:

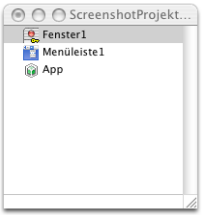
1. Wählen Sie das Objekt aus, das Sie verschlüsseln wollen und rufen Sie den Menüpunkt „Bearbeiten/Verschlüsseln“ auf. Daraufhin erscheint der Verschlüsseln-Dialog.

2. Geben Sie ein Passwort ein und bestätigen Sie dieses.

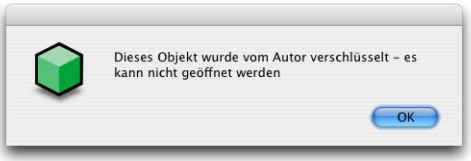
3. Wählen Sie die Option „V3 Verschlüsselung oder höher...“ nur, wenn Sie das Objekt mit REALbasic in Version 3 oder höher verwenden. Wenn Sie diese Option nicht anwählen, verwendet REALbasic die Verschlüsselung der Version 2.1. Deaktivieren Sie die Option nur, wenn Sie das Objekt mit einer REALbasic-Version kleiner V3 verwenden wollen.

Wichtiger Hinweis: Vergessen Sie nicht das Passwort!

Das Icon eines verschlüsselten Objekts erhält im Projektfenster einem kleinen Schlüssel in der rechten unteren Ecke.



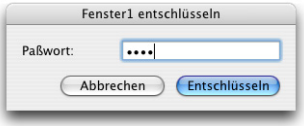
Wenn ein Programmierer (einschließlich des Autors) versucht, das Objekt zu editieren, wird folgender Dialog angezeigt.



Um das Objekt editieren zu können, müssen Sie es mit Hilfe des Passworts entschlüsseln.

Um ein Objekt zu entschlüsseln, gehen Sie folgendermaßen vor:

1. Wählen Sie das Objekt aus und rufen Sie den Menüpunkt „Bearbeiten/Entschlüsseln“ auf. Daraufhin erscheint der Entschlüsseln-Dialog.



2. Geben Sie das beim Verschlüsseln verwendete Passwort ein und klicken Sie in „Entschlüsseln“.

Nach wenigen Augenblicken verschwindet der Schlüssel im Icon des Objekts, womit angezeigt wird, dass das Objekt entschlüsselt wurde. Wenn Sie das falsche Passwort eingeben, erscheint eine Fehlermeldung.

Programmierung von Event-Handlern

REALbasic-Programme sind Event-gesteuert. Das bedeutet, dass auf Aktionen des Benutzers reagiert wird. Sie haben bereits erfahren, dass es außer den direkt vom Benutzer ausgelösten Events auch noch indirekt ausgelöste Events gibt.

Objektorientierte Programmierung

Die Programmiersprache von REALbasic ist objektorientiert. Das bedeutet, dass der Code, der bei einem Event ausgeführt wird, Teil des Objektes ist, auf das sich das Event bezieht. Dieser Code heißt Event-Handler.

Objekte können auch über eigene Methoden verfügen. Dies gestattet Ihnen, in einem Objekt Code abzulegen, auch wenn dieser Code nicht direkt mit einem Event verbunden ist. Wird in einem Fenster ein Text eingegeben, so ist es sinnvoll, den zum Sichern dieses Textes benötigten Programmcode in einer Methode dieses Fensterobjekts unterzubringen.

Weil programmierte Objekte sich etwa so verhalten sollten wie Objekte der realen Welt, sollte man Code immer dem Objekt zuordnen, zu dem er gehört. Wenn Sie zum Beispiel möchten, dass die Größe eines Fensters automatisch beim Öffnen angepasst wird, dann ist es sinnvoll, den dazu gehörenden Programmcode im Open-Event-Handler des Fensters abzulegen. Soll aber ein Knopf in diesem Fenster in den Zustand Enabled oder Disabled gesetzt werden, wenn das Fenster sich öffnet, dann gehört der Code dazu nicht in den Open-Event-Handler des Fensters, sondern in den Open-Event-Handler des Knopfes, denn dieser Codeteil betrifft nur den Knopf. Es würde genauso funktionieren, wenn Sie das anders machen, aber so ist es wirklich objektorientiert. Wollen wir doch mal ein Beispiel für die reale Welt an den Haaren herbeiziehen: Angenommen, eine Tür wird geöffnet und Sie nehmen das wahr, dann drehen Sie sich in Richtung der Tür, um festzustellen, was dort vor sich geht. Die Sensorik, die Ihnen diese Wahrnehmung ermöglicht, ist ein Teil von Ihnen und nicht ein Teil der Tür.

Der klare Vorteil der korrekten Zuordnung von Code liegt darin, dass der Code immer bei dem Objekt verbleibt. Wenn Sie es zu einem anderen Zweck nochmal woanders im Programm verwenden möchten, dann ist dafür gesorgt, dass alle Funktionen des Objekts erhalten bleiben. Fehlt ein Teil des benötigten Codes, müssen Sie ihn entweder erst aufstöbern oder neu schreiben.

Fenster (Windows-Objekt)

Events

Fenster bekommen jede Menge Events. Tabelle 8 listet die Events auf. Detailliertere Informationen finden Sie unter „Window Klasse“ in der Sprachreferenz.

Tabelle 8. Events der Window-Klasse

Event	Beschreibung
Open	Das Fenster soll geöffnet werden, wird aber noch nicht angezeigt. Auch die Steuerelemente empfangen ein Open-Event. Ein Fenster empfängt sein Open-Event, nachdem alle Steuerelemente ihr Open-Event empfangen haben. Diese Ausführungsreihenfolge sollten Sie sich merken, da sie unter Umständen eine große Rolle spielen kann.
Close	Das Fenster soll geschlossen werden, wird aber noch angezeigt. Auch die Steuerelemente empfangen ein Close-Event. Ein Fenster empfängt sein Close-Event, nachdem alle Steuerelemente ihr Close-Event empfangen haben.
CancelClose	Die Methode Quit wurde aufgerufen. Das Programm wird gleich beendet. Schickt der Event-Handler CancelClose den Wert „True“, dann wird die Quit-Methode abgebrochen und das Programm läuft weiter.
Resized	Die Größe des Fensters wurde vom Benutzer oder durch Code, der die Height- oder Width-Eigenschaft des Fensters ändert, verkleinert bzw. vergrößert.
Moved	Die Position des Fensters wurde vom Benutzer oder durch Code, der die Left- oder Top-Eigenschaft des Fensters ändert, modifiziert.
Paint	Teile des Fensters müssen neu gezeichnet werden. Das passiert beim Öffnen des Fensters, oder weil ein darüberliegendes Fenster geschlossen wurde. Diesem Event-Handler wird ein Grafikobjekt als Parameter übergeben, das die Grafik enthält, die im Fenster dargestellt wird. Grafikobjekte haben eigene Methoden, um die in ihnen enthaltene Grafik zu zeichnen. Mehr dazu unter „Graphics Klasse“ in der Sprachreferenz.
EnableMenuItems	Solange das Fenster das oberste aller Fenster ist, wird immer dieser Event-Handler aufgerufen, bevor ein Menü in der Menüleiste heruntergeklappt wird. Dies gibt Ihnen die Möglichkeit zu bestimmen, welche Menüpunkte gerade anwählbar sein sollen.

Tabelle 8. Events der Window-Klasse (Fortsetzung)

Event	Beschreibung
DropObject	Eine Datei, ein Textstück oder ein Bild wurde auf dem Fenster fallengelassen (nicht auf einem Steuerelement des Fensters, sondern auf der Fensterfläche). Dem Event-Handler wird ein Parameter übergeben, der ihm Zugriff auf dieses Objekt gibt.
KeyDown	Eine Taste, die vom Fenster behandelt werden muss, wurde gedrückt. Beispielsweise wird ein Druck auf die Tab-Taste nie an ein Steuerelement geschickt. Sie wird immer vom Fenster verarbeitet. Hat das Fenster keine Steuerelemente, die den Fokus erhalten können, wird jeder Tastendruck an das Fenster weitergegeben. Somit erzeugt dann jede Tastenbetätigung ein KeyDown-Event für das Fenster. Der übergebene Parameter entspricht dabei der gedrückten Taste.
MouseDown	Der Mausknopf wurde gedrückt, aber noch nicht losgelassen. Soll das ignoriert werden, können Sie den Event-Handler ein „False" zurückgeben lassen. Das Fenster verhält sich dann so, als ob die Maustaste nicht gedrückt worden sei. Die Mauskoordinaten (lokale Fensterkoordinaten) werden als Parameter geliefert.
MouseUp	Der Mausknopf wurde losgelassen. Die Maus war dabei noch innerhalb der Fensterfläche. Dieses Event wird nicht aufgerufen, wenn MouseDown vorher nicht „True" zurückgegeben hat, was logisch ist, denn wenn der Knopf nie gedrückt wurde, kann er ja schlecht losgelassen werden. Die Mauskoordinaten (lokale Fensterkoordinaten) werden als Parameter geliefert.
MouseDrag	Der Benutzer hat die Maus innerhalb des Fensters bewegt (die Maus befindet sich nicht über einem Steuerelement) während der Mausknopf gedrückt war. Die Mauskoordinaten (lokale Fensterkoordinaten) werden als Parameter geliefert.
MouseMove	Der Benutzer hat die Maus innerhalb des Fensters bewegt. Die Mauskoordinaten (lokale Fensterkoordinaten) werden als Parameter geliefert.
MouseEnter	Der Benutzer hat die Maus von einer Position außerhalb des Fensters in das Fenster bewegt.
MouseExit	Der Benutzer hat die Maus von einer Position innerhalb des Fensters aus dem Fenster hinaus bewegt.

Fenster öffnen

Es gibt zwei verschiedene Techniken, die Sie verwenden können, um Fenster zu öffnen. Welche Sie verwenden, hängt davon ab, was Sie mit dem Fenster vorhaben, wenn es geöffnet ist. Soll nur eine einzige Kopie des Fensters verwendet werden, dann können Sie es einfach öffnen, indem Sie eine Eigenschaft des Fensters ändern oder seine Show-Methode aufrufen.

Im folgenden Beispiel wird ein Fenster durch Zugriff auf eine Eigenschaft des Fensters geöffnet (in diesem Fall die Title-Eigenschaft (Fenstertitel) des Fensters aboutBoxWindow):

```
aboutBoxWindow.Title="Über dieses Programm"
```

Wenn keine Eigenschaften des Fensters geändert werden müssen, können Sie einfach die Show-Methode des Fensters aufrufen:

```
aboutBoxWindow.Show
```

Diese Methode funktioniert nur dann gut, wenn Sie nur eine Kopie des Fensters verwenden, da der Name des Fensters verwendet wird, um auf das Fenster zuzugreifen. Dies ist eine sehr statische Vorgehensweise. Wenn mehrere Fenster verwendet werden, dann würde REALbasic auf ein bereits geöffnetes Fenster zugreifen, statt eine weitere Kopie des Fensters zu öffnen.

Soll Ihre Anwendung mehr als eine Kopie eines Fensters öffnen, dann erzeugen Sie mit dem New-Operator eine solche Kopie. Man spricht hier von einer neuen Instanz des Fensters. Sie benötigen dazu eine Variable oder eine Eigenschaft vom Typ des Fensters. In der Variablen wird dann eine Referenz auf genau diese Kopie des Fensters angelegt, über die Sie auf das Fenster zugreifen können. Das funktioniert so:

```
Dim w as aboutBoxWindow
w=New aboutBoxWindow
```

Sie können dies verkürzen, indem Sie den New Operator in der Dim-Anweisung verwenden:

```
Dim w as New aboutBoxWindow
```

Diese einzelne Zeile öffnet einen „Über…"-Dialog.

Da aboutBoxWindow ein Objekt des Typs Window ist, können Sie auch so vorgehen:

```
Dim w as Window
w=New aboutBoxWindow
```

Das ist vor allem dann nützlich, wenn Sie mehrere verschiedene Fenster mit dem gleichen Programmcode öffnen möchten. Dies kann dann alles über dieselbe Variable stattfinden:

```
Dim w as Window
If theOptionKeyIsDown then
 w=New secretAboutBoxWindow
Else
 w=New aboutBoxWindow
End if
```

In diesem Fall könnte man mit Dim natürlich auch zwei Variablen anlegen: Eine als secretAboutBoxWindow und die andere als aboutBoxWindow. Das ist aber nicht ganz so übersichtlich und wird spätestens dann etwas mühsam, wenn Sie zehn Fenster benutzen.

Weil Fenster Objekte sind, können Sie die Variable auch als Objekt definieren, so wie hier:

```
Dim w as Object
w=New aboutBoxWindow
```

Bei Fenstern besteht normalerweise keine Notwendigkeit, diese als „Object" zu definieren. Hier ist „Window" schon etwas übersichtlicher. Bei Steuerelementen ergibt sich das schon eher. Insbesondere wenn man Steuerelemente dynamisch während des Programmablaufs erzeugen möchte. Sie können in einer Variable sogar Referenzen auf mehrere Steuerelemente (auch solche verschiedenen Typs) speichern. Siehe auch „Neue Instanzen von Steuerelementen dynamisch erzeugen" auf Seite 218. Mehr Informationen zum Gebrauch von Referenzen auf Fenster unter „Zugriff auf Bestandteile anderer Fenster" auf Seite 216.

Weitere Eigenschaften für ein Fenster anlegen

Eigenschaften sind Informationen, die ein Objekt definieren und sind mit Variablen vergleichbar, die in Event-Handlern oder Methoden verwendet werden können. Fenster haben viele vordefinierte Eigenschaften, wie beispielsweise Title, Width, Height, etc. Sie können auch eigene Eigenschaften für ein Fenster definieren, die Ihnen gestatten, für eine bestimmte Instanz eines Fensters Informationen zu speichern. Im Gegensatz zu Variablen gehören Eigenschaften einem Objekt (z.B. dem Fenster), so dass alle Event-Handler des Objekts darauf zugreifen können. Eigenschaften können sogar so deklariert werden, dass der Zugriff auch von außerhalb des Objekts möglich ist. Auf eine Variable, die innerhalb eines Event-Handlers über einen Dim-Ausdruck definiert ist, kann im Unterschied dazu von außen nicht zugegriffen werden.

Anwendungsbeispiel: In einem Textprogramm kann der Anwender mehrere Dokumente gleichzeitig öffnen und bearbeiten. Jedes Dokument wird in einem eigenen Fenster angezeigt. Beim Beenden des Programms soll der Benutzer informiert werden, falls er Änderungen an seinen Dokumenten noch nicht gespeichert hat. Das Programm muss sich also den Änderungsstatus jedes einzelnen Dokuments merken. Dies kann man elegant realisieren, indem man das Dokumentfenster mit der neuen Eigenschaft „Geändert" versieht. Dabei handelt es sich um einen Boole'schen Wert, der auf True gesetzt wird, wenn das Dokument modifiziert wurde. Schließt der Benutzer das Fenster, dann prüft der Close-Event-Handler des Fensters den Zustand der Fenstereigenschaft „Geändert". Eine solche Eigenschaft erzeugen Sie mit dem Menüpunkt **Bearbeiten/Neue Eigenschaft**.

Die Syntax zum Zugriff auf selbst definierte Eigenschaften ist identisch mit der bei den vordefinierten Eigenschaften verwendeten. Heißt das Fenster myDocumentWindow, dann wir die neue Eigenschaft wie folgt verändert:

```
myDocumentWindow.Geändert=True
```

Die Eigenschaft **Geändert** sollte nur vom jeweiligen Fensterobjekt modifiziert werden können. Deshalb sollten Sie im oben erwähnten Dialog **Neue Eigenschaft** die Checkbox **Geschützt** aktivieren (siehe Abbildung 178). Auf geschützte Eigenschaften kann nur das Objekt zugreifen, dem sie gehören.

Wenn Sie eine Eigenschaft erzeugen, können Sie sie mit einem Wert vorbelegen. Dieser wird der Eigenschaft beim Erzeugen des Fensters automatisch zugewiesen.

Der Geltungsbereich einer Eigenschaft

Für jede Eigenschaft können Sie festlegen, ob nur Programmcode, der zum selben Objekt gehört, darauf zugreifen darf oder ob auch andere Methoden und Event-Handler auf die Eigenschaft Zugriff erhalten. Folgende Geltungsbereiche sind möglich:

Öffentlich (ohne Einschränkungen): Auf eine öffentliche Eigenschaft kann überall im Projekt zugegriffen werden, außerhalb des Objekts mittels Punkt-Notation (z.B. *Fenster1.Titel)*.

Geschützt (nur aktuelles Fenster und Unterklassen): Auf eine geschützte Eigenschaft kann nur von den Event-Handlern und Methoden des Fensters und seiner Steuerelemente zugegriffen werden. Wenn Sie versuchen, außerhalb des Fensters auf die Eigenschaft zuzugreifen, erhalten Sie eine Fehlermeldung.

Privat (nur aktuelles Fenster): Eine private Eigenschaft verhält sich wie eine geschützte Eigenschaft. Allerdings kann nicht von Unterklassen des Fensters darauf zugegriffen werden. Ein Fenster, dass von einem anderen Fenster abgeleitet wurde, verfügt nur über dessen öffentliche und geschützte Eigenschaften.

Eine neue Eigenschaft für ein Fenster legen Sie so an:

1. Öffnen Sie im Code-Editor den Code des entsprechenden Fensters.
2. Wählen Sie **Bearbeiten/Neue Eigenschaft**.
3. Geben Sie den Namen der Eigenschaft ein und definieren Sie ihren Datentyp. Heißt die Eigenschaft **Geändert** und soll vom Typ Boolean sein, dann tippen Sie *Geändert As Boolean* ein.

Abb. 178: Deklarieren einer Eigenschaft

4. Sie können der Eigenschaft einen Vorgabewert zuweisen. Dazu verwenden Sie folgende Syntax:

   ```
   EigenschaftName AS Datentyp = Wert
   ```

   Falls es sich bei der Eigenschaft um ein Integer-Array mit vier Elementen handeln soll, schreiben Sie folgendes:

   ```
   WindowParameters(3) as Integer
   ```

   Sie können auch ein leeres Array als Eigenschaft deklarieren. Dazu geben Sie das Array mit leeren Klammern an:

   ```
   WindowParameters() as Integer
   ```

5. Wählen Sie einen Gültigkeitsbereich aus. Außerdem können Sie durch Aktivieren der Checkbox **Im Eigenschaftenfenster zeigen** erreichen, dass die Eigenschaft im Eigenschaftenfenster des Fenster-Objekts erscheint, so dass Sie ihr auch dort einen Wert zuweisen können (und nicht nur mittels Programmcode).
6. Klicken Sie „OK“ und speichern die Eigenschaft.
7. Im Code-Editor können Sie Notizen zu der Eigenschaft eingeben. Der eingegebene Text ist nicht ausführbar, auch wenn er gültigen REALbasic-Code enthält. Sie können der Eigenschaft hier keine Werte zuweisen. Dies können Sie in den Methoden und Event-Handlern des Fensters oder – falls es eine öffentliche Eigenschaft ist – auch in Methoden und Event-Handlern anderer Objekte.

Ist die Eigenschaft geschützt oder privat, wird ihr Icon im Browser-Bereich entsprechend gekennzeichnet.

Ändern einer Eigenschaft

1. Öffnen Sie den Code-Editor für das Fenster, das die zu ändernde Eigenschaft enthält.
2. Klappen Sie im Browser die Liste der Eigenschaften des Fensters auf.
3. Führen Sie einen Doppelklick auf die Eigenschaft aus oder selektieren Sie sie mit Einfachklick und rufen dann **Bearbeiten/Bearbeiten** (⌘-E bzw. Strg-E) auf.

Abb. 179: „Eigenschaft ändern“-Dialog

4. Führen Sie alle notwendigen Änderungen durch und klicken in „OK“.

Löschen einer Eigenschaft:

1. Öffnen Sie den Code-Editor für das Fenster, das die zu löschende Eigenschaft enthält.
2. Klappen Sie im Browser die Liste der Eigenschaften des Fensters auf.
3. Klicken Sie einmal auf die zu löschende Eigenschaft, um sie zu selektieren.
4. Wählen Sie **Bearbeiten/Löschen** oder Kontextmenü/Löschen.

Eigenschaften eines Fensters können vom Quellcode des Fensters oder seiner Steuerelemente über ihren Namen angesprochen werden. Der Name des Fensters ist dazu nicht nötig, wie das folgende Beispiel zeigt:

```
Title="Ein neues Fenster"
```

Ist kein Fenstername angegeben, wird immer der Name des gegenwärtigen Fensters angenommen. Wenn Sie wollen, können Sie immer mit der Self-Funktion auf die Eigenschaften des Fensters verweisen. Wenn Self im Event-Handler eines Steuerelements verwendet wird, verweist es auf das Elternobjekt, welches in diesem Fall das Fenster ist, in dem sich das Steuerelement befindet. Der Ausdruck

```
Self.Title="Mein neues Fenster"
```

verweist also ebenso auf die Title-Eigenschaft des Elternfensters.

Konstanten in einem Fenster anlegen

Eine Konstante entspricht einer Eigenschaft, die über ihre gesamte Lebensdauer einen konstanten Wert enthält. Wenn Sie eine Konstante erzeugen, weisen Sie ihr einen Wert zu, der später nicht verändert werden kann. Sie können eine Konstante auch in Ihrem Code erzeugen. Auch diese kann nicht durch eine Zuweisung geändert werden.

Sie können Konstanten für Fenster, Module und Klassen erzeugen. Sie könne auch lokale Konstanten innerhalb einer Methode erzeugen.

Der Gültigkeitsbereich von Konstanten

Eine Fenster-Konstante besitzt genau wie eine Eigenschaft einen Gültigkeitsbereich, der festlegt, welche Teile eines Projekts auf sie zugreifen können:

Öffentlich: Die Konstante kann innerhalb des gesamten Projekts verwendet werden. Um die Konstante innerhalb des Objekts zu verwenden, das sie deklariert hat, müssen Sie nur ihren Namen angeben, z.B. „Konstantenname".

Außerhalb des Objekts müssen Sie die Punkt-Notation verwenden. Zum Beispiel „Objektname.Konstantenname".

Geschützt: Geschützte Konstanten können nur innerhalb der Methoden und EventHandler des Fensters oder dessen Steuerelemente verwendet werden.

Privat: Eine private Konstante entspricht einer geschützten mit dem Unterschied, dass sie nicht „vererbt" werden kann. Abgeleitete Fenster enthalten nur öffentliche und geschützte Konstanten.

Lokalisieren einer Applikation mit Hilfe von Konstanten

Wenn Sie mehrere Versionen Ihrer Applikation für verschiedene Plattformen und Sprachen erzeugen müssen, sollten Sie alle Textausgaben über Konstanten definieren. REALbasic erlaubt es Ihnen, einer Konstante mehrere verschiedene Werte zuzuweisen, die abhängig von der aktuellen Zielplattform und -sprache beim Compilieren eingesetzt werden.

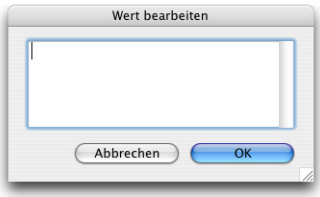
Dazu verwenden Sie die Tabelle im „Neue Konstante"-Dialog. Dort bestimmen Sie, welchen Wert die Konstante für eine bestimmte Sprache und Plattform hat. Häufig wird es nützlich sein, den Gültigkeitsbereich global zu halten, damit alle Teile des Projekts darauf zugreifen können. Dies ist allerdings nur für Module möglich. Wenn Sie Text-Konstanten an einer Stelle sammeln, können Sie sich Ärger bei der Pflege dieser Konstanten ersparen.

Dieser Vorgang wird detailliert im Abschnitt über Module beschrieben, aber er weicht nicht vom Vorgehen für eine Fenster-Konstante ab. Wenn Sie Konstanten verwenden wollen, um Fenster zu lokalisieren, folgen Sie den Ausführungen im Abschnitt „Eine Konstante einem Modul hinzufügen" auf Seite 232.

Um eine Konstante einem Fenster zuzuweisen, gehen Sie folgendermaßen vor:

1. Öffnen Sie den Code-Editor für das Fenster, dem Sie eine Konstante hinzufügen wollen.
2. Wählen Sie nun den Menüeintrag **Bearbeiten/Neue Konstante**. Daraufhin erscheint der „Neue Konstante"-Dialog.

3. Bestimmen Sie die Bezeichnung, den Typ, den Gültigkeitsbereich und den Wert der Konstanten. Falls Sie mehrzeiligen Text eingeben wollen, klicken Sie auf den kleinen Button neben dem Eingabefeld, um den Text in einem separaten Dialog einzugeben:



4. Benutzen Sie – falls erforderlich – die Lokalisierungstabelle, um der Konstanten für unterschiedliche Plattformen und Sprachen passende Werte zuzuweisen.
5. Klicken Sie „OK“ um die Konstante zu speichern.

Methoden in einem Fenster anlegen

Fenster können auch eigene Methoden erhalten. Das hat den Vorteil, dass Code, der nur für ein bestimmtes Fenster verwendet werden soll, auch mit dem Fenster zusammen abgelegt werden kann. Wird in einem Fenster ein Dokument angezeigt, das der Benutzer ändern kann, dann ist es sinnvoll den Programmteil, der sich um das Sichern des Dokumentes kümmert, auch mit im Fenster unterzubringen. Wenn Sie also eine Methode **ÄnderungenSichern** im Fenster ablegen, dann ist dieser Code im Fenster enthalten, wenn Sie ein solches Fenster später in einem anderen Projekt benötigen.

Parameter für Methoden angeben

Mit Hilfe von Parametern können Sie Werte an eine Methode übergeben. Diese können von der Methode verwendet werden und sogar modifiziert und zurück gegeben werden.

Parameter werden wie Eigenschaften deklariert (z.B. *Alter as Integer*). Um festzulegen, dass eine Methode Parameter annimmt, geben Sie deren Bezeichnung und Datentyp an. Möchten Sie mehrere Parameter übergeben, trennen Sie diese mit einem Komma. Wenn eine Methode zum Beispiel zwei Parameter vom Typ Integer und String annehmen soll, würden Sie die Parameter folgendermaßen deklarieren:

```
myInt as Integer, myString as String
```

Parameter werden innerhalb der Methode wie lokale Variablen behandelt. Die Methode kann den Wert eines Parameters ändern, wie bei lokalen Variablen auch. *myInt* verhält sich also wie eine Integer-Variable, *myString* wie ein String.

Es können auch Arrays als Parameter übergeben werden. Dazu fügen Sie dem Parameternamen zwei leere Klammern an. Wenn Sie zum Beispiel ein Integer-Array mit vier Elementen an eine Methode übergeben können wollen, deklarieren Sie den Parameter folgendermaßen:

```
myInt() as Integer
```

Da die Deklaration nicht die Größe des Arrays vorschreibt, kann ein Array beliebiger Größe übergeben werden. Sie können die Anzahl der Elemente in einem Array mit Hilfe der Ubound-Funktion ermitteln. Wollen Sie einer Methode ein Array übergeben, geben Sie einfach den Array-Namen ohne Klammern an.

Sie können auch mehrdimensionale Arrays übergeben. Auch wenn Sie die Anzahl der Elemente einer Dimension nicht angeben müssen, so müssen Sie doch die Anzahl der Dimensionen angeben. Dies tun Sie, indem Sie Anzahl der Dimensionen minus eins Kommata zwischen die Klammern schreiben. Wenn zum Beispiel aNames ein zweidimensionales Array ist, wäre die Deklaration folgendermaßen:

```
aNames(,) as String
```

Es gibt noch eine weitere Möglichkeit, eine Reihe von Parametern an eine Methode zu übergeben. Sie können das ParamArray-Schlüsselwort verwenden. Das ParamArray-Schlüsselwort gibt an, dass eine unbestimmte Anzahl von Werten

eines bestimmten Typs übergeben wird. Wenn Sie eine Methode schreiben, die eine Liste von Integer-Werten akzeptieren soll, würden Sie die Parameter so deklarieren:

```
ParamArray nums as Integer
```

Wenn Sie die Methode aufrufen, können Sie eine Liste von Integer-Werten übergeben, die durch ein Komma getrennt sind (als wären sie separate Parameter). Zum Beispiel:

```
myMethod(5,7,2,10)
```

Innerhalb der Methode ist `nums` ein Integer-Array, welches die Werte enthält.

Rückgabewerte von Methoden

Eine Methode kann auch einen Wert zurückgeben. Solche Methoden werden Funktionen genannt. Sie können eine Funktion genau so erzeugen wie eine Methode. Der einzige Unterschied ist, dass Sie den Datentyp des Rückgabewerts bestimmen müssen.

Der Rückgabewert einer Funktion kann ein Array oder ein einzelner Wert sein. Wenn Sie ein Array zurückgeben wollen, fügen Sie der Bezeichnung des Datentyps einfach leere Klammern an. Wenn Sie beispielsweise ein Array von Integer-Werten zurückgeben wollen, schreiben Sie „Integer()“. Für ein mehrdimensionales Array geben Sie Anzahl der Dimensionen minus eins Kommata innerhalb der Klammern an. Zum Beispiel „Double(,)“ für ein zweidimensionales Double-Array.

Damit eine Funktion einen Wert zurück liefert, verwenden Sie das Schlüsselwort „Return“. Werden in Ihrer Funktion beispielsweise zwei Integer-Arrays addiert und das Ergebnis in einer Variablen Total abgelegt, können sie dieses mit

```
Return Total
```

an den aufrufenden Code zurück liefern. Bitte beachten Sie, dass mit diesem Ausdruck die Funktion verlassen wird. Code, der sich hinter diesem Ausdruck befindet, wird nicht ausgeführt.

Der Gültigkeitsbereich von Methoden

Wenn Sie eine Methode erzeugen, legen Sie auch fest, welche Teile des Projekts diese Methode erreichen können, um sie auszuführen. Sie können festlegen, ob die Methode nur innerhalb des Fensters und seiner Methoden und Event-Handler erreichbar ist, oder aus dem gesamten Projekt erreicht werden kann. Für den Gültigkeitsbereich gibt es verschiedene Möglichkeiten:

Öffentlich: Die Methode kann innerhalb des gesamten Projekts verwendet werden. Dies wäre zum Beispiel notwendig, damit ein MenuHandler des App-Objekts auf die Methode zugreifen kann. Um die Methode innerhalb des Objekts zu verwenden, das die Methode deklariert, greifen Sie über den Methodennamen auf sie zu. Außerhalb verwenden Sie die Punkt-Notation (Fenster1.Methode).

Geschützt: Geschützte Methoden können nur in dem Objekt, in dem sie deklariert wurden oder in abgeleiteten Klassen verwendet werden. Sie würden eine gschützte Methode verwenden, wenn diese mit Informationen des Fensters arbeitet und Sie nicht wünschen, dass diese von außen aufgerufen werden kann. Die Methode wird über ihren Namen aufgerufen. Wenn Sie eine solche Methode außerhalb ihres Elternobjekts aufrufen, wird eine Fehlermeldung anzeigt.

Privat: Private Methoden können nur in dem Objekt aufgerufen werden, in dem sie deklariert wurden. Auch abgeleitete Klassen können nicht auf solche Methoden zugreifen, sondern enthalten nur öffentliche oder geschützte Methoden der übergeordneten Klasse. Sie rufen private Methoden über ihren Namen auf. Wenn Sie eine solche Methode außerhalb des Elternobjekts aufrufen, wird eine Fehlermeldung angezeigt.

So legen Sie eine neue Methode an:

1. Öffnen Sie den Code-Editor mit dem Code des entsprechenden Fensters.
2. Wählen Sie **Bearbeiten/Neue Methode**.

3. Geben Sie den Namen der Methode ein, legen Sie die Parameter fest und definieren Sie den Rückgabetyp, falls die Methode einen Wert zurückliefern soll.

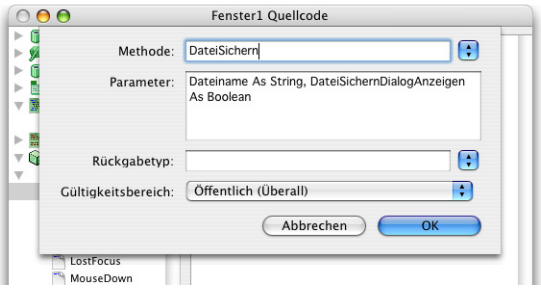
   Achten Sie darauf, dass Sie bei der Benennung der Methode kein reserviertes Wort verwenden. Verwenden Sie die Felder für die Parameter und den Rückgabetyp dazu, den Datentyp jedes Parameters und des zurückgegebenen Werts zu deklarieren. Mehrere Parameter werden durch Komma voneinander getrennt (Beispiel: Alter as Integer, Name as String).

   Das Popup-Menü rechts vom Rückgabetyp-Feld enthält die gebräuchlichsten Datentypen. Aber Sie können auch jeden anderen gültigen Datentyp im Rückgabetyp-Feld-Feld angeben.

   Für das Parameter-Feld gibt es eine Reihe weitergehender Optionen. Weitere Informationen finden Sie in den Abschnitten „Einen Parameter als Wert oder Referenz übergeben“ auf Seite 213, „Vorgabewerte für Parameter“ auf Seite 214, „Setter Methods“ auf Seite 215, „Zugriff auf Bestandteile anderer Fenster“ auf Seite 216 und „Konstruktoren und Destruktoren“ auf Seite 216.

4. Wählen Sie nun den Gültigkeitsbereich der Methode aus. Weitere Informationen zu diesem Thema finden Sie im vorherigen Abschnitt. Eine vollständige Methoden-Deklaration sieht beispielsweise so aus:

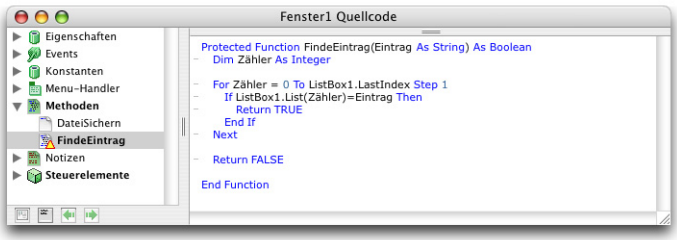
Abb. 180: Dialogbox „Neue Methode“



5. Wenn Sie damit fertig sind, klicken Sie auf **OK**, um die Methoden-Deklaration zu speichern.

   Sobald Sie **OK** geklickt haben, öffnet sich ein Editor-Fenster, in das Sie den Quelltext der eben deklarierten Methode eingeben können. Achten Sie darauf, dass die erste Zeile die Deklaration Ihrer Methode als Sub oder Function enthält. Innerhalb des Editors können Sie an dieser Deklaration nichts ändern. Führen Sie im Browser einen Doppelklick auf den Methodennamen aus, wenn Sie die Deklaration bearbeiten wollen.

   Wenn Sie die Methode als geschützt oder privat deklariert haben, wird dies im „Sub“- oder „Function“-Ausdruck angezeigt, wie in der Abbildung zu sehen ist. Wenn es eine öffentliche Methode ist, steht keine zusätzliche Bezeichnung im „Sub“- oder „Function“-Ausdruck. Das Icon einer geschützten oder privaten Methode enthält ein Warndreieck.



Ändern des Namens, der Parameter oder des Rückgabewerts einer Methode:

1. Öffnen Sie den Code-Editor für das Fenster, das die zu ändernde Methode enthält.
2. Klappen Sie im Browser die Liste der Methoden des Fensters auf.

3. Doppelklicken Sie auf die Methode, oder wählen Sie sie durch einen einfachen Mausklick aus und rufen anschließend **Bearbeiten/Bearbeiten** (⌘-E oder Strg-E) auf, um die Methode zu modifizieren.

Löschen einer Methode:

1. Öffnen Sie den Code-Editor für das Fenster, das die zu löschende Methode enthält.
2. Klappen Sie im Browser die Liste der Methoden des Fensters auf.
3. Klicken Sie einmal auf die zu löschende Methode, um sie zu selektieren.
4. Wählen Sie **Löschen** aus dem Bearbeiten- oder dem Kontextmenü oder drücken Sie Backspace.

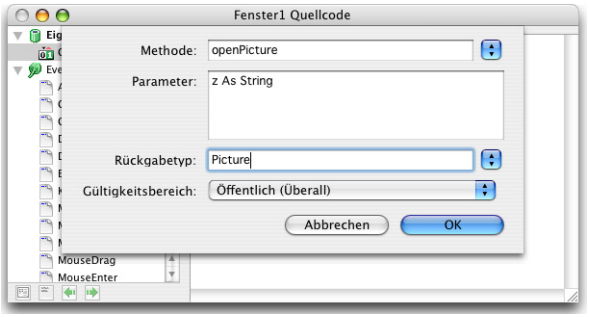
Eine Beispiel-Methode

Folgendes Beispiel zeigt, wie man eine Methode schreibt und wie man sie aus einem Event-Handler im Fenster aufruft.

Nehmen wir an, Ihr Programm muss dem Anwender die Möglichkeit bieten, Bilder zu öffnen und die Bilder unterschiedlichen Objekten zuzuweisen. Sie können eine einfache Methode verwenden, um die Datei zu öffnen und das Bild zurückzuliefern. Der Event, der die Methode aufruft, kann dann das Objekt einem Interface-Objekt zuweisen.

Die Methode bekommt den Namen der zu öffnenden Datei als Parameter übergeben und liefert ein Picture-Objekt. Daher wird sie als Funktion deklariert.

Abb. 181: Die OpenPicture-Deklaration



Die Methode besteht aus folgendem Code (die Funktions-Deklaration kann nur über die **Methode bearbeiten**-Dialogbox geändert werden).

```
Function openPicture (s as String) as Picture
  Dim f as folderItem
  Dim p as picture
  f=GetFolderItem(s)
  if f.exists then
    p=f.openAsPicture
  else
    MsgBox "Ungültige Datei!"
  end if
  Return p
End Function
```

Mit dieser Methode können Sie dann Bilddateien öffnen, indem Sie ihr einfach den Pfad der Datei als String übergeben. Folgende Zeile zeigt z.B. das Bild in einem ImageWell-Steuerelement an, indem das Bild der Image-Eigenschaft des ImageWell zugewiesen wird.

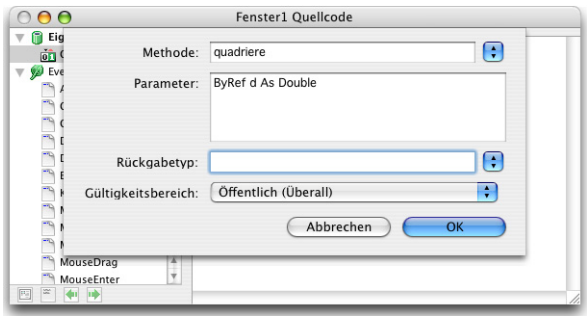
```
Imagewell1.image=openPicture("Hintergrundbild")
```

Einen Parameter als Wert oder Referenz übergeben

Wenn Sie die Parameter der Methode definieren, können Sie zwischen zwei Parametertypen wählen: Übergabe als Wert (by value) oder als Referenz (by reference). Die Voreinstellung ist Übergabe als Wert. Bei dieser Art erhält die Methode den Wert als Parameter und kann mit diesem natürlich Operationen ausführen. Der Parameter wird wie eine lokale Variable behandelt. Sie müssen nichts Besonderes machen, um Parameter als Wert zu übergeben. Die Übergabe als Wert ist im vorangegangenem Beispiel gezeigt worden: Ein String-Parameter, der einen Pfadnamen enthält, wurde an die Methode übergeben und von dieser als Parameter in einem anderen Funktionsaufruf verwendet.

Wenn Sie Übergabe als Referenz verwenden, übergeben Sie einen Verweis auf den Parameter, also einen Zeiger (Pointer). Wenn die Methode einen Parameter als Referenz bekommt, kann es den Wert des Parameters nach Belieben verändern und den geänderten Wert nach Beenden der Methode an Sie zurückliefern. Folgende Methoden-Deklaration verwendet einen Parameter, der als Referenz übergeben wird.

Abb. 182: Einen Parameter als Referenz übergeben



Das Schlüsselwort **ByRef** in der Parameter-Deklaration bewirkt, dass der Parameter als Referenz und nicht sein Wert übergeben wird. Die Methode selbst besteht nur aus folgender Zeile:

```
d=d*d
```

Die Methode ändert also den Wert des übergebenen Parameters. Solch eine Methode kann folgendermaßen (z.B. im Action-Event eines PushButtons) aufgerufen werden:

```
Sub Action()
  Dim Label as String
  Dim i as Double
  If Editfield1.text <> "" then
    i=Val(Editfield1.text)
    Label="Das Quadrat von "+EditField1.text+" ist "
    Quadriere(i) //Übergabe als Referenz
    StaticText1.Text=Label+Str(i)+"."
  Else //kein Wert eingegeben
    Beep
    MsgBox "Bitte geben Sie in die Textbox eine Zahl ein."
  End if
End Sub
```

Wenn Sie diese Methode ausführen, werden Sie bemerken, dass sich der Wert von **i** nach dem Aufruf von **Quadriere** geändert hat. Das ist nicht möglich, wenn man ihn als Wert (by value) übergibt.

Sie können auch Arrays als Referenz übergeben. Dazu müssen Sie nichts weiter angeben, da Arrays grundsätzlich als Referenz übergeben werden. Es reicht also aus, wenn Sie einen Parameter als Array deklarieren:

```
aNames() as String
```

Vorgabewerte für Parameter

Wenn Sie für ein Methode die Parameter deklarieren, können Sie wahlweise für jeden Parameter einen voreingestellten Wert angeben. Dies erreichen Sie, indem Sie in der **Neue Methode**-Dialogbox die Deklaration als Zuweisung schreiben. Wenn Sie z.B. den Parameter a als Integer deklarieren möchten, würden Sie in das Parameterfeld der **Neue Methode**-Dialogbox folgendes schreiben:

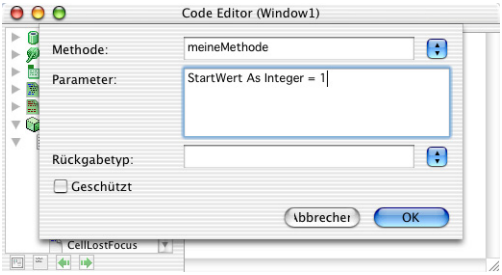
```
a As Integer
```

Um dem Parameter den Vorgabewert 10 zuzuweisen, würden Sie folgendes eingeben:

```
a As Integer = 10
```

Schauen Sie sich einmal die untenstehende Methoden-Deklarierung an:

Abb. 183: Einem Parameter einen Vorgabewert zuordnen



Normalerweise müssen Sie der Methode *meineMethode* bei jedem Aufruf einen Integer-Wert übergeben. Wenn Sie einen Vorgabewert angeben, können Sie den Parameter im Aufruf weglassen. Wenn Sie den Vorgabewert benutzen wollen, lassen Sie den Parameter einfach aus:

```
meineMethode
```

In diesem Beispiel wird nun der Vorgabewert von 1 verwendet.

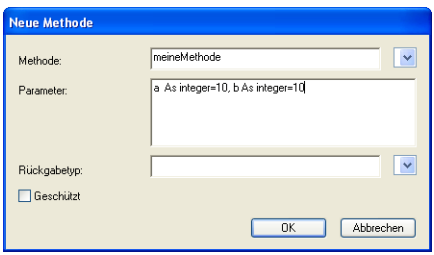
Wenn Sie einen anderen Wert verwenden wollen, übergeben Sie diesen als ganz normalen Parameter:

```
meineMethode(10)
```

In diesem Fall wird *meineMethode* mit dem übergebenen Wert 10 arbeiten und den Vorgabewert ignorieren.

Sie können auch für mehrere Parameter Vorgabewerte angeben. So ist auch der folgende Ausdruck gültig:

Abb. 184: Zwei Parametern Vorgabewerte zuordnen



In diesem Fall können Sie die Methode folgendermaßen aufrufen:

```
meineMethode
```

Nun werden die beiden Vorgabewerte eingesetzt. Wenn Sie den Vorgabewert ersten Parameters überschreiben wollen, übergeben Sie nur einen Wert:

```
meineMethode(100)
```

In diesem Beispiel wird der Wert 100 als Parameter a übergeben. Zum Überschreiben des zweiten Parameters müssen Sie zwei Parameter übergeben. Geben Sie dazu einfach den voreingestellten Wert des ersten Parameters an.

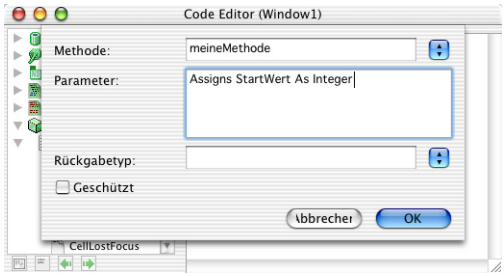
„Setter Methods"

Sie können auch mit dem Zuweisungs-Operator einen Wert an eine Methode übergeben. Um z.B. den Wert 10 an eine Methode zu übergeben, können Sie folgenden Ausdruck verwenden:

```
meineMethode=10
```

Den Zuweisungs-Operator können Sie allerdings nur einsetzen, wenn Sie das Schlüsselwort „Assigns" beim Deklarieren der Methode verwenden.

Abb. 185: Verwenden des Schlüsselworts „Assigns" beim Deklarieren von Methoden

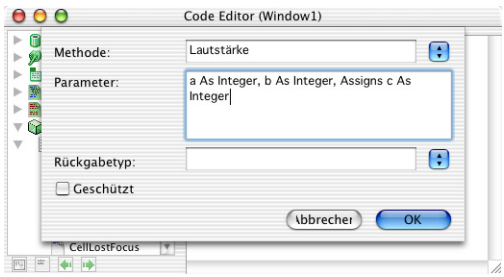


Wenn *meineMethode* wie im obigen Beispiel deklariert wird, dann können Sie ihr auf folgende Art und Weise einen Wert übergeben:

```
meineMethode = 10
```

Sie können das „Assigns"-Schlüsselwort auch bei Methoden einsetzen, die mehr als einen Parameter erwarten. Dann müssen Sie „Assigns" mit dem letzten Parameter verwenden. Sie können „Assigns" nur für einen Parameter pro Methode einsetzen. Die folgende Deklaration einer Methode ist also korrekt:

Abb. 186: Das Schlüsselwort „Assigns" mit mehr als einem Parameter verwenden



Zum Aufruf dieser Methode verwenden Sie die folgende Syntax:

```
Lautstärke(5,4)=10
```

Wenn Sie das „Assigns" Schlüsselwort wie im obigen Beispiel einsetzen, können Sie die „normale" Syntax nicht mehr verwenden:

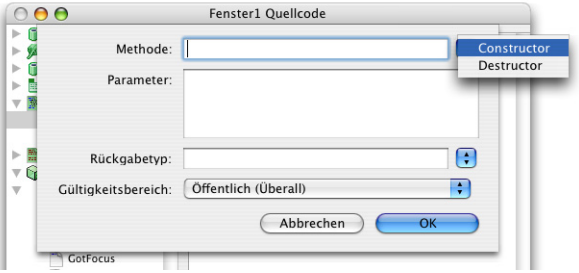
```
Lautstärke(5,4,10)//funktioniert nicht
```

Konstruktoren und Destruktoren

Ein Konstruktor ist eine Methode, die abläuft, wenn das Objekt, zu dem sie gehört, zum ersten Mal ins Leben gerufen wird. Normalerweise verwenden Sie einen Konstruktor, um ein Objekt zu initialisieren. Ein weiteres Beispiel für einen Konstruktor wäre eine Methode, die die Rahmen- und Hintergrundfarbe eines Canvas-Objekts festlegt und initial ein Objekt im Canvas-Objekt zeichnet.

Ein Destruktor ist eine Methode, die abläuft, wenn ein Objekt gelöscht wird oder verschwindet, wie z.B. beim Schließen eines Fensters.

Mit Hilfe des Code-Editors können Sie ganz einfach Konstruktor- und Destruktor-Methoden erstellen. Anstatt einen Namen für die Methode einzugeben, verwenden Sie einfach das Popup-Menü rechts vom Namenseingabefeld in der **Neue Methode**-Dialogbox. Hier haben Sie die Auswahl zwischen Konstruktor und Destruktor. Im Anschluss an Ihre Wahl erscheint der korrekte Name des Konstruktors oder Destruktors im Namenseingabefeld der Methode.



Ein Fenster (oder jedes beliebige Objekt) kann nur einen Konstruktor und einen Destruktor besitzen. Wenn Sie bereits einen Konstruktor für ein Objekt geschrieben haben, bietet das PopupMenu nur den Destruktor zur Auswahl an und umgekehrt.

Zugriff auf Bestandteile anderer Fenster

Objekte in anderen Fenstern können verwendet werden, indem Sie den Namen des Fensters angeben, in dem sich das betreffende Objekt befindet, und dahinter den Namen des Objekts. Selbstverständlich geht das nur, wenn der Gültigkeitsbereich des Objekts *öffentlich* ist.

Sie verwenden die Syntax *Fenstername.Objektname*. Wird in Fenster1 ein Knopf geklickt, der dann die Find-Methode aufrufen soll, die sich im Fenster2 befindet, dann macht man das so:

```
Fenster2.Find "Hallo"
```

Die Eigenschaften anderer Fenster können ebenso verwendet werden. Wieder wird in Fenster1 ein Knopf geklickt. Diesmal ändert er die Title-Eigenschaft von Fenster2:

```
Fenster2.Title="Hallo Fenster 2"
```

Im Falle eines Steuerelements kann der Bezeichnung des Steuerelements wiederum die Bezeichnung einer Eigenschaft folgen. Nehmen wir an, ein Button im Fenster1 würde bei einem Klick den Text eines Textfelds in Fenster2 ändern. Die Syntax sähe so aus:

```
Fenster2.StaticText1.Text="NeuerText"
```

Also *Fenstername.Elementname.Eigenschaft*.

Die Beispiele würden sich allerdings immer auf die erste Instanz von Fenster2 beziehen, egal wieviele Kopien des Fensters geöffnet sind.

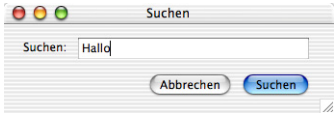
Soll von einem Fenster mehr als eine Kopie verwendet werden, so müssen Sie irgendwo eine Referenz auf die Fenster ablegen, damit das Programm immer weiß, auf welches Fenster sich eine Aktion beziehen soll. Wo Sie das tun, hängt ganz von der konkreten Anwendung ab. Angenommen, Sie haben mehrere Instanzen eines Fensters namens „DocWin-

dow“ geöffnet, in denen jeweils der Text eines Dokuments angezeigt wird. Ein Knopf in diesem Fenster öffnet ein Suchenfenster, in dem ein Suchbegriff eingegeben werden kann. Die Suche soll in der Instanz des Fensters durchgeführt werden, in der der Suchen-Knopf gedrückt wurde. Da es mehrere Dokumentfenster geben kann, muss dem Suchenfenster mitgeteilt werden, in welchem Fenster gesucht werden soll, was man mit Hilfe einer Referenz auf das Fenster erledigt. Zu diesem Zweck legen wir eine Eigenschaft an und nennen diese „Zielfenster“. Die Eigenschaft wird im Suchenfenster angelegt (denn das Suchenfenster benötigt diese Information) und als Typ für diese Eigenschaft verwenden wir DocWindow. Nehmen wir mal an, dass es nur erlaubt ist, ein einziges Suchenfenster geöffnet zu haben (was man zum Beispiel dadurch erreicht, dass man das Suchenfenster modal macht), dann sieht die Syntax so aus:

```
SuchenFenster.Ziel=Self
```

Die Funktion Self liefert eine Referenz auf das Objekt selbst, von dem aus sie aufgerufen wurde. Da diese Zeile vom Suchen-Knopf in der entsprechenden Instanz des Dokumentenfensters aufgerufen wird, liefert Self also eine Referenz auf genau diese Instanz des Fensters und speichert sie in der Eigenschaft Ziel des Suchenfensters. Drückt der Benutzer dann später im Suchenfenster den Knopf „Finden“, kann der Code über die Referenz in der Ziel-Eigenschaft auf den Text im Dokumentfenster zugreifen und darin suchen.

Abb. 187: Ein Fenster zum Suchen



In der Abbildung hat das Fenster ein EditField, das wir „Suchbegriff“ nennen. Hier gibt der Benutzer den zu suchenden Begriff ein. Nehmen wir an, dass das Dokumentenfenster über eine Methode namens Suchen verfügt, die einen Suchbegriff als Parameter erhält und diesen dann im Dokument sucht und eine Fundstelle hervorhebt. Dann muss der Suchen-Knopf im Suchenfenster nur noch sehr wenig tun. Er muss nämlich nur der Suchen-Methode des Dokumentfensters mitteilen, was zu suchen ist und diese aufrufen:

```
Ziel.Suchen Suchbegriff.Text
```

Die Eigenschaft Ziel, die dem Suchenfenster gehört, enthält eine Referenz auf das Dokumentfenster, das das Suchenfenster aufgerufen hat. Somit kann man diese Referenz prima verwenden, um die Suchen-Methode dieses Fensters aufzurufen. Als Parameter wird die Text-Eigenschaft des EditFields mit dem Namen „Suchbegriff“ übergeben.

Die Eigenschaft „Ziel“ bzw. die darin enthaltene Referenz auf ein Fenster kann auch dazu verwendet werden, um Eigenschaften von Steuerelementen im Fenster zu verändern. Der Suchen-Knopf könnte zum Beispiel auf den Zustand „disabled“ gesetzt werden, solange das Suchenfenster angezeigt wird:

```
Ziel.Suchknopf.Enabled=False
```

In unserem Beispiel wurde der Eigenschaft Ziel des Suchenfensters der Typ DocWindow zugeordnet. Soll das Suchenfenster auch mit anderen Fenstern verwendbar sein, dann ist es geschickter, diesem den Typ Window zu geben. Das macht den Programmcode aber wiederum schlechter lesbar, da man dann nicht mehr aus dem Typ der Eigenschaft darauf schließen kann, dass sie sich auf DocWindow bezieht. Verwenden Sie den allgemeinen Typ Window also nur dann, wenn dies aus praktischen Gründen erforderlich erscheint.

Steuerelemente

Steuerelemente sind Objekte, die innerhalb eines Fensters erscheinen und die mit eigenem Code auf Ereignisse reagieren können. Sie können keine selbstprogrammierten Eigenschaften oder Methoden erhalten, wie es bei Fenstern möglich ist. Dies geht nur dann, wenn Sie eine neue Klasse anlegen, die von einem bestehenden Steuerelement abgeleitet ist.

Dazu erstellen Sie zuerst eine neue Klasse, die auf einem der Steuerelemente basiert, und fügen sie dem Projektfenster hinzu. Als SuperClass legen Sie Control fest und statten anschließend Ihre Klasse mit eigenen Methoden, Eigenschaften oder Event-Handlern aus. Um einem Fenster eine Instanz dieser Klasse hinzuzufügen, draggen Sie sie vom Projektfenster auf einen Fenstereditor. Ein ausführlicheres Beispiel erhalten Sie im Abschnitt „Was ist eine Unterklasse?“ auf Seite 305.

Events

Steuerelemente, wie beispielsweise Fenster, empfangen Events und haben Event-Handler, die darauf reagieren können.

Für jedes Event, auf das ein Steuerelement reagieren kann, gibt es einen entsprechenden Event-Handler. Die Sprachreferenz enthält eine vollständige Liste aller Events, auf die Steuerelemente reagieren können. Wenn Sie die Sprachreferenz lesen, sollten Sie im Hinterkopf behalten, dass jedes Steuerelement Eigenschaften und Events von seinem Eltern-Objekt erbt. So leiten sich zum Beispiel die meisten Steuerelemente von der RectControl-Klasse ab und haben demzufolge alle Events und Eigenschaften der RectControl-Klasse.

Neue Instanzen von Steuerelementen dynamisch erzeugen

Es gibt Situationen, in denen Sie nicht das gesamte Benutzer-Interface von vorneherein festlegen können, sondern einzelne Steuerelemente erst zur Laufzeit erzeugt werden müssen. Dies ist mit REALbasic möglich, sofern das Fenster, in dem das geschehen soll, bereits ein Steuerelement desselben Typs enthält. Das existierende Steuerelement kann dann als Vorlage für das zu erzeugende Element verwendet werden. Wollen Sie beispielsweise einen PushButton erzeugen, dann muss in dem Fenster ein Knopf vorhanden sein, den Sie klonen können. Da man alle Steuerelemente über die Eigenschaft “Visible” auch unsichtbar in einem Fenster ablegen kann, ist dies keine besondere Einschränkung.

Nehmen wir an, Sie möchten den „PushButton1“ klonen, der bereits im Fenster vorhanden ist.

So legt man zur Laufzeit eine Kopie des Steuerelements an:

1. Öffnen Sie das Eigenschaftenfenster von PushButton1 und setzen Sie den Index auf Null.

   Wenn neue Steuerelemente zur Laufzeit erzeugt werden, liegen sie in einem Array von Steuerelementen.

2. In dem Action-Event von PushButton1 reservieren Sie mit der Dim-Anweisung Platz für einen weiteren PushButton.
3. Weisen Sie nun der Variablen eine neue Instanz des Steuerelements mit Hilfe des New-Operators zu. Dabei geben Sie die Bezeichnung des zu klonenden Steuerelements an. In unserem Fall ist das PushButton1. Der Action-EventHandler von PushButton1 sähe dann so aus:

```
Dim pb as PushButton
pb= New Pushbutton1  //Klonen von PushButton1
pb.Caption="Clone"  //Beschriftung des Klons ändern, um Verwechselungen zu vermeiden
pb.Left=me.Left+me.Width+10  //Den Klon ein Stück nach rechts verschieben
```

4. Wählen Sie nun **Debug/Programm** starten.

Ihr Programm erzeugt bei einem Klick auf den Button „Original“ einen Klon. Die Caption-Eigenschaft der Variablen pb enthält die entsprechende Beschriftung.

Wenn Sie den Button Klon anklicken, erzeugen Sie rechts daneben einen weiteren Klon, und so fort.

Da alle Steuerelemente, die Sie zur Laufzeit erzeugen, den Code mit dem Ursprungsobjekt teilen, ist es wichtig, dass man die einzelnen Steuerelemente zur Laufzeit unterscheiden kann. Dazu können Sie die Eigenschaft Index an den Action-EventHandler übergeben. Nähere Informationen erhalten Sie im folgenden Abschnitt „Gemeinsamer Code für ein Array aus Steuerelementen“ auf Seite 219.

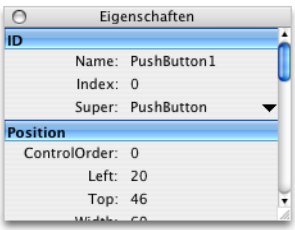
Wenn Sie zur Laufzeit unterschiedliche Steuerelemente erzeugen müssen, dafür aber nur eine Variable verwenden wollen, wählen Sie den Typ für die Variable, von dem sich die zu erzeugenden Steuerelemente ableiten. Wenn Sie zum Beispiel einen RadioButton und einen CheckButton erzeugen wollen, wählen Sie den Typ RectControl für die Variable, da beide Button-Typen davon abgeleitet sind. Beachten Sie aber, dass Sie in dem Fall nicht auf die Eigenschaften eines CheckButton oder RadioButton zugreifen können. Um herauszufinden, welche übergeordnete Klasse sich die Steuerelemente teilen, beachten Sie deren Einträge in der Sprachreferenz.

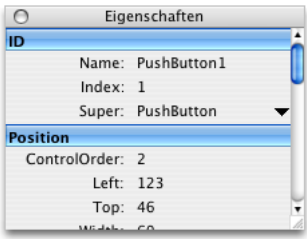
Gemeinsamer Code für ein Array aus Steuerelementen

Haben Sie mehrere Steuerelemente desselben Typs, die den selben Code verwenden sollen, legen Sie am besten ein Array mit den Steuerelementen (Control Array) an. In einem Array aus Steuerelementen können zwei oder mehr Steuerelemente auf den gleichen Code zurückgreifen. Ein Array aus Steuerelementen erzeugen Sie, indem Sie allen Steuerelementen des Arrays denselben Namen geben (Name-Eigenschaft). Verwenden Sie dann die Index-Eigenschaft, um die einzelnen Elemente des Control Arrays zu identifizieren. Geben Sie einem Steuerelement einen Namen, der bereits verwendet wurde, fragt REALbasic, ob Sie ein Array mit diesen Steuerelementen anlegen möchten. Zum Beispiel nennen Sie einen RadioButton „RB1“. Wenn Sie nun einen weiteren RadioButton „RB1“ nennen, wird REALbasic Sie fragen, ob Sie ein Control-Array erzeugen wollen.

Abb. 188: Ein Array aus Steuerelementen erzeugen

Wenn Sie nun „Ja“ klicken, bekommt das erste Objekt den Index 0 und das Objekt, welches Sie gerade umbenennen, den Index 1.





Beachten Sie, dass die beiden Steuerelemente denselben Namen tragen und sich hauptsächlich durch die Eigenschaft Index unterscheiden. Weitere Steuerelemente mit demselben Namen werden anhand der Eigenschaft Index durchnumeriert.

Haben Sie eine Checkbox namens „Option“ angelegt und erzeugen eine zweite Checkbox, die Sie ebenfalls „Option“ nennen, dann fragt REALbasic nach, ob Sie ein Array aus Steuerelementen erzeugen wollen. Die erste Checkbox erhält dann den Index 0 und die zweite den Index 1.

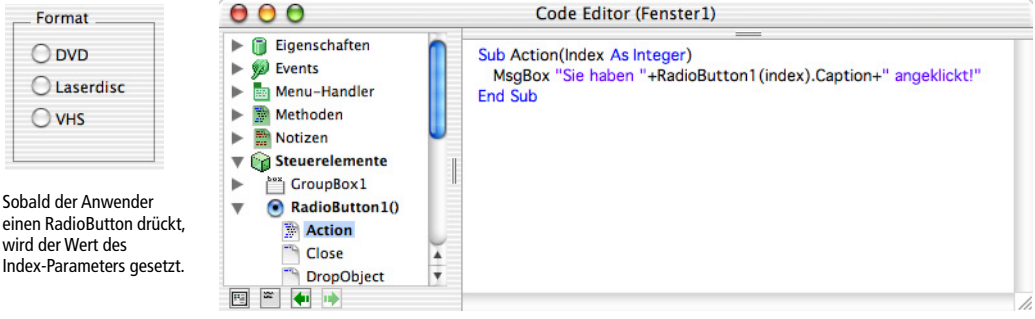
Alternativ können Sie Null als Wert der Index-Eigenschaft beim ersten Steuerelement eingeben und dann den Namen des ersten Steuerelements dem zweiten zuweisen. REALbasic weist dann automatisch der Index-Eigenschaft des zweiten Steuerelements 1 zu und so weiter.

Im Code-Editor erscheint dann nur noch ein Steuerelement dieses Namens. Hinter dem Namen erscheinen Klammern, die anzeigen, dass es sich um ein Array handelt. Diese Steuerelemente werden nun alle vom gleichen Code gesteuert. Jedem Event wird die Index-Nummer des Steuerelements übergeben, so dass Sie immer feststellen können, auf welches Steuerelement sich die Aktion gerade bezieht.

Im folgenden Beispiel gehören alle drei RadioButton-Steuerelemente zu einem Control Array. Alle drei Steuerelemente besitzen den Namen „rb1“ und werden über ihre Index-Eigenschaft unterschieden. Im Code-Editor wird der Wert der Index-Eigenschaft automatisch an jede Methode übergeben. Im Action-Event-Handler stellen Sie z.B. fest, welcher der RadioButtons angeklickt wurde.

Beachten Sie im folgenden Beispiel, dass der Index-Parameter automatisch zur Zeile der Methoden-Deklaration hinzugefügt wird. Anstelle eines einfachen „Sub“ steht hier ein „Sub Action(Index As Integer)“. Beim Parameter „Index“ handelt es sich um die Index-Eigenschaft aus dem Eigenschaftenfenster.

Abb. 189: Ein Control Array und sein Action-Event-Handler



Durch einen einfache Case-Befehl können Sie feststellen, welcher Knopf gedrückt wurde und entsprechend reagieren. Das folgende Beispiel testet ein Control-Array mit drei RadioButton:

```
Sub Action(Index as Integer)
       Select Case Index
        Case 0
        msgBox "you selected radio button "+str(index)
        Case 1
        msgBox "you selected radio button "+str(index)
        Case 2
        msgBox "you selected radio button "+str(index)
        End select
End Sub
```

Ersetzen Sie die msgBox-Aufrufe durch den Code, den Sie für den jeweiligen Button ausführen wollen.

Drag & Drop

Drag & Drop ist ein wichtiges Element im Benutzer-Interface vieler Programme. REALbasic unterstützt Drag & Drop für Text, Bilder, Dokumente und bestimmte Datentypen.

Wenn etwas mit der Maus gezogen wird, wird ein DragItem-Objekt erzeugt. DragItem-Objekte verfügen über eine Text-Eigenschaft, die verwendet wird, um gedraggten Text aufzunehmen, eine Picture-Eigenschaft ein gedraggtes Bild, eine MacData-Eigenschaft für bestimmte Datentypen und eine FolderItem-Eigenschaft, die auf ein gedraggtes Dokument, einen Ordner oder ein Anwendungsprogramm verweist. Manchmal müssen Sie selbst die Daten dort ablegen. In anderen Fällen sind diese dort automatisch vorhanden.

Ein DragItem kann mehr als ein Objekt enthalten und die Objekte müssen nicht unbedingt den gleichen Typ haben. Wenn Sie dem Anwender erlauben, solche Objekte zu draggen, müssen Sie zusätzliche Objekte innerhalb des DragItem-Objekts erzeugen und beim „Drop“ alle Objekte des DragItems berücksichtigen.

DragItems, die auf den Schreibtisch oder andere Anwendungsprogramme gezogen werden, verhalten sich wie erwartet: Ein Text, der auf den Schreibtisch gezogen wird, erzeugt eine Clipping-Datei. Ist das DragItem ein Bild, wird entsprechend eine Clipping-Datei mit diesem Bild erzeugt.

Dragging von Text aus einem EditField

Nur der Text in einem EditField, Zeilen einer ListBox, Teile eines Canvas-Steuerelements, Bilder eines ImageWells und Fenster können mit der Maus gezogen werden. Das klingt vielleicht wie eine drastische Einschränkung, ist es aber in der Praxis nicht. Diese Steuerelemente beherbergen nämlich die einzigen Datentypen, mit denen andere Programme, die Drag & Drop unterstützen, etwas anfangen können.

Der Text in einem EditField kann automatisch gezogen werden, ohne dass spezieller Programmieraufwand dazu nötig ist, vorausgesetzt, dass die Eigenschaft MultiLine des EditField aktiviert wurde. Dann wird automatisch ein DragItem-Objekt erzeugt und der Text landet in der Text-Eigenschaft des DragItem-Objekts.

Dragging einer Zeile Text aus einer ListBox

Damit der Benutzer eine Zeile aus einer ListBox ziehen kann, muss die EnableDrag-Eigenschaft der ListBox aktiviert werden. Beginnt der Benutzer eine Zeile aus einer ListBox zu ziehen, dann wird das DragRow-Event der ListBox aufgerufen. Diesem werden als Parameter das DragItem übergeben und die Nummer der Zeile, die gezogen wird. Sie müssen dann die Text-Eigenschaft des DragItem definieren. Da das DragRow-Event eine Funktion ist, müssen Sie „True“ zurückliefern, damit das Draggen überhaupt stattfinden kann. Geben Sie „False“ oder keine Daten zurück, dann wird das Draggen abgebrochen. Hier ein Beispiel für einen entsprechenden Event-Handler:

```
Function DragRow(Drag as DragItem, Row as Integer)
 Drag.Text=Me.List(Row)- chr(13)//Text ins DragItem schreiben
 Return True //Dragging erlauben
End Function
```

Dragging aus einem ImageWell

Drag&Drop in oder aus einem ImageWell ist einfach. Wenn Sie aus einem ImageWell draggen wollen, müssen Sie:

- ein DragItem-Objekt mit der NewDragItem-Methode erzeugen,
- die Daten, die gedraggt werden sollen, in die neue DragItem-Instanz laden,
- die Drag-Methode des DragItems aufrufen, um das Dragging zu erlauben.

Das DragItem-Objekt ist der Container, der das gedraggte Bild enthält. NewDragItem ist eine Methode, die als Parameter die Ausmaße des Rechtecks entgegennimmt, die der Anwender sieht, wenn er mit dem Draggen beginnt. Die Drag-Methode wird aufgerufen, wenn die zu draggenden Daten in das DragItem geladen wurden. Diesen Code platzie-

ren Sie in den MouseDown-Event-Handler des ImageWell, da der Anwender die Maustaste zu Beginn des Dragvorgangs drückt.

```
Function MouseDown(X as Integer,Y as Integer) As Boolean
 Dim d as DragItem
 d=Me.NewDragItem(Me.left,Me.top,Me.width,Me.height)
 d.picture=Me.image
 d.Drag //Erlaube Draggen
End Function
```

Dragging aus einem Canvas-Steuerelement

Das Draggen des Backdrop-Images aus einem Steuerelement ist identisch mit dem Draggen des Bildes aus einem ImageWell. Sie platzieren den Code in den MouseDown-Event-Handler des Canvas-Steuerelements, da der Anwender das Dragging beginnt, indem er die Maustaste drückt.

```
Function MouseDown(X as Integer,Y as Integer) As Boolean
 Dim d as DragItem
 d=NewDragItem(Me.Left, Me.Top, Me.Width, Me.Height)
 d.Picture=Me.Backdrop //liefert DragItem die Daten
 d.Drag //Erlaube Dragging
End Function
```

Wenn Sie das Drop programmieren, müssen Sie die Picture-Eigenschaft des DragItem einer Eigenschaft des Steuerelements zuweisen, das den Drag erhält.

Dropping

Damit der Benutzer auf einem Fenster ein DragItem, das er mit der Maus gezogen hat, ablegen kann, muss das Fenster signalisieren, dass es bereit ist, Daten auf diese Weise entgegenzunehmen. Es gibt vier Methoden, die jedes Steuerelement aufrufen kann, um die Art der Daten, die es entgegennimmt, zu definieren. Sie können angeben, dass das Steuerelement oder Fenster mehr als einen Datentyp akzeptiert, indem Sie so viele Methoden der folgenden Tabelle verwenden wie nötig:

Tabelle 9. Methoden die den Datentyp steuern, der auf ein Steuerelement gezogen werden kann

Name	Beschreibung
AcceptTextDrop	Definiert, dass das Steuerelement Text entgegennimmt.
AcceptPictureDrop	Definiert, dass das Steuerelement Bilder entgegennimmt.
AcceptFileDrop (Type)	Definiert, dass das Steuerelement oder Fenster Dateien der angegebenen Typen entgegennimmt. Die Dateien müssen im Dateitypen-Dialog des Projekts definiert sein.
AcceptMacDataDrop (Type)	Definiert, dass das Steuerelement oder Fenster Dateien des durch den vierbuchstabigen Type festgelegten Datentyps entgegennimmt, z.B. AcceptMacDataDrop("mytp"). Verwenden Sie dies, um Daten eines bestimmten Formats zu draggen/droppen oder festzulegen, wohin Textstrings gedroppt werden können.

Normalerweise ruft das Steuerelement oder das Fenster im Open-Event-Handler eine oder mehrere dieser Methoden auf. Sind bestimmte Voraussetzungen nötig, damit ein Drag & Drop entgegengenommen wird, dann kann dies auch an anderer Stelle geschehen. In den meisten Fällen wird beim Fallenlassen eines DragItems der DropObject-Event-Handler des Objektes aufgerufen, auf dem das DragItem fallengelassen wurde. Das DragItem wird dem Event-Handler überge-

ben. Hat das Objekt über die eben aufgeführten Methoden zuvor definiert, dass nur bestimmte Datentypen akzeptiert werden, kann Ihr Unterprogramm die Daten direkt aus dem DragItem auslesen. Die Datentypen dazu sind folgende:

Tabelle 10. DragItem-Eigenschaften, die den Objekt-Typ anzeigen, der auf ein Steuerelement gezogen wurde

Name	Beschreibung
FolderItem	Repräsentiert ein Anwendungsprogramm, einen Ordner oder ein Dokument, das fallengelassen wurde.
Picture	Enthält ein Bild, falls eines fallengelassen wurde.
Text	Enthält einen Text, falls einer fallengelassen wurde.
MacData (Type)	Macintosh-Daten des Typs, der durch den vierbuchstabigen Code angegeben wird. Daten dieses Formats werden als ein String zurückgeliefert.
PrivateMacData (Type)	Daten (des angegebenen Typs), die gedraggt werden. Type ist ein vierbuchstabiger Resource Type oder vom Programmierer definierter vierbuchstabiger Code. Dieser Dateityp kann nicht auf eine andere Anwendung gedraggt werden. Wird nur auf dem Macintosh unterstützt.

Können mehrere Datentypen entgegengenommen werden, muss der DropObject-Event-Handler zunächst herausfinden, welche Art von Daten fallengelassen wurde. Dazu bietet DragItem folgende Funktionen:

Tabelle 11. DragItem-Funktionen, die den Datentyp ermitteln

Name	Beschreibung
FolderItemAvailable	Liefert „True", wenn ein oder mehrere Anwendungsprogramme, Ordner oder Dokumente fallengelassen wurden.
PictureAvailable	Liefert „True", wenn ein Bild fallengelassen wurde.
TextAvailable	Liefert „True", wenn ein Text fallengelassen wurde.
MacDataAvailable (Type)	Liefert „True", wenn Macintosh-Daten des durch den vierbuchstabigen Type-Code angegebenen Typs gedroppt wurden.

Dropping von Objekten auf EditFields

Text, der auf mehrzeilige EditFields gedroppt wird, wird automatisch an der Stelle der Einfügemarke platziert. Der DropObject-Event-Handler des EditFields wird nicht aufgerufen. Bilder und Dateien, die auf ein mehrzeiliges EditField gedroppt werden, lösen jedoch den DropObject-Event-Handler aus. Wenn Sie z.B. eine Textdatei in ein EditField droppen wollen und der Inhalt im EditField erscheinen soll, müssen Sie den Inhalt der durch das FolderItem des DragItem-Objekts spezifizierten Datei einlesen und in das Editfield einfügen.

Im folgenden Beispiel wurde ein EditField darauf vorbereitet, gedroppte Text-Dateien entgegenzunehmen. Me ist die allgemeine Schreibweise, um sich auf das Objekt zu beziehen, zu dem der Event-Handler gehört:

```
Sub DropObject(Obj as DragItem)
 If Obj.FolderItemAvailable Then
 Obj.FolderItem.OpenStyledEditField Me
 End If
End Sub
```

Da mehr als eine Datei gleichzeitig fallengelassen werden könnte, müssen Sie mit der NextItem-Funktion prüfen, ob eventuell noch eine weitere Datei folgt. Die NextItem-Funktion ändert außerdem die FolderItem-Eigenschaft des DragItem auf die nächste Datei. Ändert man das letzte Beispiel entsprechend ab, sieht es so aus:

```
Sub DropObject(Obj as DragItem)
 If Obj.FolderItemAvailable Then
 Do
  Obj.FolderItem.OpenStyledEditField Me
 Loop Until Not Obj.NextItem
 End If
```

```
End Sub
```

Objekte auf ListBoxen ablegen

Wenn Sie Text auf eine ListBox draggen wollen, müssen Sie die Listbox entsprechend präparieren, indem Sie folgende Zeile in ihren Open-Event-Handler platzieren (oder in einen anderen Event-Handler, der vor der Drag&Drop-Aktion aufgerufen wird):

```
Me.AcceptTextDrop
```

Danach müssen Sie der ListBox mitteilen, welche Aktion mit dem gedraggten Text ausgeführt werden soll. Dies machen Sie in deren DropObject-Event-Handler:

Folgender DropObject-Event-Handler stellt fest, ob das gedraggte Objekt Text enthält. Ist dies der Fall, erzeugt er eine neue Zeile und weist dieser die Text-Eigenschaft des gedraggten Objekts zu.

```
Sub DropObject (Obj as DragItem)
 If Obj.TextAvailable then
  Me.AddRow(obj.text)
 end if
End Sub
```

Dropping von Objekten auf ImageWells und Canvas Steuerelemente

Um dem Anwender das Dropping eines vom Schreibtisch gedraggten Bildes oder PICT-Dokuments zu erlauben, fügen Sie folgende Befehle in den Open-Event-Handler des ImageWell- oder Canvas-Steuerelements ein:

```
Me.AcceptPictureDrop
Me.AcceptFileDrop("image/x-pict")
```

Der zweite Befehl setzt voraus, dass Sie Ihrem Projekt über **Bearbeiten/Dateitypen** den PICT-Dateityp hinzugefügt haben.

Danach muss man dem ImageWell oder Canvas mitteilen, was es mit dem jeweiligen DragItem machen soll. Dies geschieht im DropObject-Event-Handler. Folgendes funktioniert bei einem ImageWell:

```
Sub DropObject (Obj as DragItem)
 If Obj.PictureAvailable then
  ImageWell1.Image=Obj.Picture
 Elseif Obj.FolderItemAvailable then
  me.image=Obj.FolderItem.OpenAsPicture
 End if
End Sub
```

Zunächst wird getestet, ob das DragItem ein Bild oder eine Datei (FolderItem) ist. Handelt es sich um ein Bild, wird die Image-Eigenschaft des DragItems der Image-Eigenschaft des ImageWell zugewiesen. Wenn es ein FolderItem ist, wird die PICT-Datei eingelesen und der Image-Eigenschaft des ImageWells zugewiesen.

Wenn Sie das gedroppte Objekt der BackDrop-Eigenschaft eines Canvas-Steuerelements zuweisen wollen, ist der DropObject-Handler praktisch identisch:

```
If Obj.PictureAvailable then
 Canvas1.Backdrop=Obj.Picture
Elseif Obj.FolderItemAvailable then
 Canvas1.Backdrop=Obj.FolderItem.OpenAsPicture
End if
```

Sie können auch ein Text-Objekt auf ein Canvas-Steuerelement draggen und die DrawString-Methode der Graphics-Klasse verwenden, um den Text zu zeichnen. Um das Draggen zu erlauben, verwenden Sie folgende Zeile in einem Event-Handler, der vor dem Drag & Drop des Anwenders aufgerufen werden muss:

```
me.AcceptTextDrop
```

Als nächstes überprüfen Sie im DropObject-Event-Handler das Objekt auf Text. Folgender Code akzeptiert gedraggten Text und schreibt diesen an die angegebene Stelle:

```
If Obj.TextAvailable then
 Canvas1.Graphics.DrawString(obj.text,20,20,50)
End if
```

Sie werden die Daten durch Verwenden des Paint-Event-Handler aktualisieren müssen, wenn Sie möchten, dass die Daten von Dauer sind.

MacData und PrivateMacData Eigenschaften

Die MacData und PrivateMacData-Eigenschaften erlauben Drag&Drop von anderen Datentypen. Jede Eigenschaft verwendet einen vierbuchstabigen Type, der dem vierbuchstabigen Resource-Code des Formats der Daten entspricht, die Sie Draggen möchten. Wenn Sie z.B. das Dragging von Sound-Clips unterstützen wollen, müssten Sie als vierbuchstabigen Code „snd " verwenden.

Unabhängig vom Format werden die Daten im DragItem als String gespeichert und der DropObject-Event-Handler würde diese Daten einer Eigenschaft übergeben, die einen String erwartet. Das Steuerelement oder Fenster, das das DragItem empfängt, muss in der Lage sein, mit diesem Format umzugehen. In den meisten dieser Fälle handelt es sich um benutzerdefinierte Steuerelemente, die Toolbox-Aufrufe und ein Plugin verwenden, das die Daten verarbeitet.

Um dem Anwender das Dropping einer „snd "-Resource auf ein Steuerelement zu erlauben, müssten Sie folgendes in den Open-Event-Handler des Steuerelements schreiben:

```
acceptMacDataDrop("snd ")
```

Der DropObject-Event-Handler würde die übergebenen Daten an eine Eigenschaft übergeben, die einen String speichert. Um den Sound abzuspielen, müsste das Steuerelement Toolbox-Aufrufe ausführen oder die Daten einem Plugin übergeben, da REALbasic keine Möglichkeit bietet, Sound-Daten an eine Sound-Klasse zu übergeben.

Sie können MacData oder PrivateMacData auch verwenden, um internes Drag&Drop zu realisieren. Wenn Sie einen Format-Typ verwenden, der sich vom Namen von jedem Resource-Typ unterscheidet, können Sie steuern, welche Objekte auf ein Steuerelement gedroppt werden dürfen. Z.B. verwendet der DragRow-Event-Handler einer ListBox den Typ „mytp", um das Format zu definieren:

```
Function DragRow(drag as DragItem, row as Integer) as Boolean
 Drag.MacData("mytp")=me.List(row)
 Return True
End Function
```

Das DragItem wird der Zeile in der ListBox zugewiesen, in die es gedraggt wurde. Obwohl die Zeile ein normaler String ist, kann dieses DragItem nicht auf ein Steuerelement gedroppt werden, so lange dieses nicht Daten des Typs „mytp" akzeptiert. Auf diese Weise können Sie z.B. ein Steuerelement erzeugen, das nur DragItems eines bestimmten Typs akzeptiert und gedraggte Objekte aus dem Finder, anderen Programmen oder anderen Steuerelementen abweist.

Die ListBox, die die Daten erhält, würde folgenden Befehl verwenden, um das Dropping der Daten zu erlauben:

```
me.AcceptMacDataDrop("mytp")
```

Der DropObject-Event-Handler weist die Daten einer neuen Zeile zu:

```
If obj.MacDataAvailable("mytp") then
 me.AddRow(obj.MacData("mytp"))
end
```

Die PrivateMacData-Eigenschaft arbeitet in der gleichen Art, außer dass das DragItem nicht vom Schreibtisch oder einer anderen Anwendung kommen kann.

MacData und PrivateMacData sind nur in Macintosh-Applikationen verfügbar.

Menüs und Menüpunkte

Menüs und Menüpunkte werden ähnlich wie Steuerelemente behandelt und sind genauso objektorientiert. Dies bedeutet, dass Menüs vom Application-Objekt, von Fenstern oder von Steuerelementen beeinflusst werden können. Beim Erstellen eines neuen Projekts legt REALbasic automatisch ein Menubar-Objekt namens MenuBar1 an. Dabei handelt es sich um die voreingestellte globale Menüleiste für die gesamte Anwendung.

Sie können Ihrem Projekt weitere Menüleisten hinzufügen und Menüleisten bestimmten Fenstern zuordnen. Unter Windows und Linux ist die Menüleiste einem Fenster fest zugeordnet und tritt immer mit dem entsprechenden Fenster in Erscheinung. Auf dem Mac wird beim Aktivieren eines Fensters die aktuelle Menüleiste durch die des Fensters ersetzt.

Selektiert der Benutzer einen Menüpunkt oder drückt er dessen zugeordnetes Tastenkürzel, dann wird ein Event ausgelöst, genau als ob er einen PushButton gedrückt hätte. Bei Menüs heißt der Event-Handler Menu-Handler.

Code für einen Menüpunkt

Ein Menu-Handler wird so eingegeben:

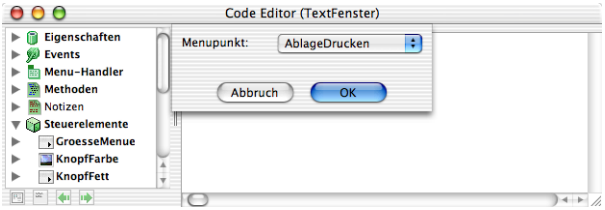
1. Öffnen Sie den Code-Editor für dieses Fenster oder die Klasse.
2. Wählen Sie **Bearbeiten/Neuer Menu-Handler**.

   Die Dialogbox „Neuer Menu-Handler« erscheint.
3. Wählen Sie einen Menüpunkt aus dem Popup-Menü mit den Menüpunkten.

   Existiert der Menüeintrag nicht, da er dynamisch erzeugt wird, tragen Sie dessen Bezeichnung ein.
4. Klicken Sie **OK**.

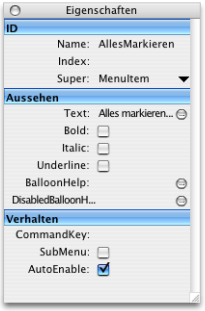
   Der Code-Editor erscheint mit ausgewähltem Menu-Handler.
5. Geben sie den Code ein, der ausgeführt werden soll, wenn dieser Menüpunkt aufgerufen wird.

Abb. 190: Die Dialogbox „Neuer Menu-Handler“



Menüpunkte aktivieren

Jeder Menüpunkt besitzt eine Autoenable-Eigenschaft, die den Wert TRUE oder FALSE annehmen kann. Steht diese Eigenschaft auf TRUE, bleibt der Menüpunkt aktiviert, bis Sie ihn explizit deaktivieren. Autoenable können Sie im Eigenschaftenfenster setzen.



Der voreingestellte Wert ist TRUE. Solange Sie die Autoenable-Eigenschaft nicht deaktivieren, wird der Menüpunkt automatisch aktiviert. Sie sollten die Autoenable-Eigenschaft für Menüpunkte benutzen, die ständig aktiv sein sollen. Dabei könnte es sich z.B. um einen Menübefehl handeln, der ein neues Dokument erzeugt oder bestehendes öffnet.

Wenn Sie die Autoenable-Eigenschaft abschalten, ist der Menüeintrag per Voreinstellung inaktiv. Sie haben nun die Verantwortung, den Menüpunkt immer dann zu aktivieren, wenn die Umstände es erfordern. Dies erreichen Sie über das EnableMenuItems-Event. Es bietet sich an, die Autoenable-Eigenschaft auszuschalten und das EnableMenuItem einzusetzen, falls ein Menüpunkt nur unter ganz bestimmten Bedingungen aktiviert werden soll. Wenn beispielsweise ein „Save"-Menüpunkt nur dann aktiv sein soll, wenn der Benutzer seit dem letzten Speichervorgang Änderungen an seinem Dokument vorgenommen hat, dann würden Sie zu diesem Zweck das EnableMenuItems-Event einsetzen, um zu bestimmen, wann der Menüpunkt aktiviert oder deaktiviert werden soll.

Klickt der Benutzer auf ein Menü, um daraus einen Menüpunkt auszusuchen, wird ein EnableMenuItems-Event ausgelöst. Dies gibt Ihnen die Gelegenheit zu entscheiden, ob die Menüpunkte im gewählten Menü in der gegenwärtigen Situation selektierbar sein sollen oder nicht. REALbasic überprüft zuerst, ob das Steuerelement, das gerade den Fokus hat, eine Menübehandlung vornehmen kann. Ist dies der Fall, wird dort ein EnableMenuItems-Event ausgelöst. Dann bekommt das oberste geöffnete Fenster ein solches Event (vorausgesetzt ein Fenster ist geöffnet) und am Schluss wird es an das Application-Objekt geschickt.

Menüpunkte sind Objekte wie Steuerelemente auch. Daher besitzen Sie die Eigenschaft Enabled, die anzeigt, ob der Menüpunkt gerade gewählt werden kann oder nicht. Der folgende EnableMenuItems-Event-Handler prüft z.B. eine Eigenschaft „Geändert", um zu entscheiden, ob der Menüpunkt **Sichern** aktiviert sein soll:

```
Sub EnableMenuItems()
 If Me.Geändert Then
 AblageSichern.Enabled=True
 End If
End Sub
```

Behandeln der Menüpunkte mit Steuerelementen

Hat ein Steuerelement den Fokus und ist in der Lage, Menüpunkte zu verarbeiten, dann wird sein EnableMenuItems-Event-Handler ausgeführt. Ist ein Menüpunkt selektierbar und wird vom Benutzer angewählt, dann wird der Menu-Handler des Steuerelements für den gewählten Menüpunkt aufgerufen, sofern das Steuerelement einen solchen Menu-Handler hat. Damit ein Steuerelement Menüpunkte verarbeiten kann, muss es den Fokus erhalten können (auf einem Macintosh muss es ein EditField, ListBox, ComboBox oder Canvas sein) und es muss auf einer Klasse aufsetzen, die Sie selbst definiert und im Projekt angelegt haben. Es ist nicht bei Steuerelementen möglich, die Sie aus der Steuerelementepalette auf das Fenster gezogen haben. Mehr Informationen darüber finden Sie im Kapitel „Wiederverwendbare Objekte durch Klassen" auf Seite 304.

Behandeln der Menüpunkte, wenn ein Fenster geöffnet ist

Sie wissen bereits, dass der EnableMenuItems-Event-Handler des obersten Fensters aufgerufen wird, wenn der Benutzer einen Menüpunkt selektieren will, bevor der EnableMenuItems-Event-Handler des Application-Objekts aufgerufen wird. Dies gibt Ihnen die Gelegenheit, zunächst zu ermitteln, ob die Bedingungen für das oberste Fenster dazu führen sollten, um bestimmte Menüpunkte zugänglich zu machen.

Behandeln von Menüpunkten, wenn keine Fenster offen sind

Sind keine Fenster offen, wird das EnableMenuItems-Event an das Application-Objekt geschickt. Wird ein Menüpunkt ausgewählt, dann wird der Menu-Handler im Application-Objekt aufgerufen, sofern ein passender definiert ist.

Beim Anlegen eines neuen Projekts wird immer eine auf dem Application-Objekt beruhende Klasse dem Projekt hinzugefügt. Diese Klasse nennt sich App-Klasse.

Abb. 191: Die App-Klasse im Projektfenster eines Standard-Projekts

Für den Fall, dass der Endbenutzer alle Fenster Ihrer Anwendung schließt und sich dann dazu entschließt, über einen Menüpunkt ein neues Fenster zu erzeugen, müssen Sie diesen Menüpunkt im Application-Objekt aktivieren, da Ihnen zu diesem Zeitpunkt keine Fenster zur Verfügung stehen, um den Menüpunkt zu aktivieren und die Menüauswahl zu verarbeiten (vorausgesetzt Sie verzichten auf die AutoEnable-Eigenschaft des Menüpunktes). Öffnen Sie dazu im Projektfenster den Code-Editor der App-Klasse, klappen Sie die Events aus und wählen Sie das EnableMenuItems-Event.

Neue Menüpunkte dynamisch erzeugen

Dies funktioniert so ähnlich wie die dynamische Erzeugung von Steuerelementen. Ein Menüpunkt, der als Vorlage dienen kann, muss bereits existieren. Dieser wird dann kopiert. Sie können die Eigenschaften der Kopie dann ändern (zum Beispiel den Text, der im Menü stehen soll). Menüpunkte müssen immer einen Indexwert in der Index-Eigenschaft haben, um als Vorlage benutzt werden zu können. Weisen Sie der Index-Eigenschaft dieses Vorlagemenüpunkts eine 0 zu, um ein Array mit Menüpunkten zu erzeugen. Die Menu-Handler bekommen dann einen Indexwert, der Ihnen jeweils erlaubt festzustellen, welcher Menüpunkt angeklickt wurde. Ohne den Indexwert können Sie das nicht. Kopien der Vorlage können Sie mit dem New-Operator erzeugen. Im Beispiel wird ein neuer Menüpunkt angelegt, der aus der Vorlage „WindowItem“ hervorgeht:

```
Dim t as MenuItem
t=New WindowItem
```

Denken Sie daran, dass Sie nach dem Erzeugen des Arrays mit den Menüpunkten die einzelnen Elemente wie bei einem normalen Array ansprechen. Um den ersten Menüpunkt zum Beispiel selektierbar zu machen (dies ist Element 0 aus dem Beispiel mit WindowItem) gehen Sie so vor:

```
WindowItem(0).Enabled=True
```

Wenn Sie programmgesteuert Menüpunkte entfernen möchten, die Sie zuvor erzeugt haben, müssen Sie sich die Referenz auf den Menüpunkt merken, die Sie erhalten, wenn der Menüpunkt erzeugt wird. Mit der Close-Methode und diesem Parameter kann der Menüpunkt dann entfernt werden. Wenn Sie die Referenzen im obigen Beispiel in einem Array namens WindowRefs gespeichert haben, dann können Sie ein bestimmtes Element entfernen, indem Sie beispielsweise so vorgehen:

```
WindowRefs(4).Close
```

Klassen

Klassen können verwendet werden, um benutzerdefinierte Steuerelemente zu entwerfen, die auch auf Benutzereingaben reagieren können. Mehr dazu finden Sie im Abschnitt „Anlegen selbstdefinierter Steuerelemente mit Klassen“ auf Seite 322.

Kapitel 6 Globale Funktionen durch Module

Objektorientierte Programmierung ist sehr effizient, aber manchmal werden Sie Methoden, Funktionen oder Eigenschaften benötigen, die nicht an ein Objekt gebunden sind. Dies könnte zum Beispiel eine finanzmathematische Berechnung sein, die an verschiedenen Stellen des Programms benötigt wird. Dazu müssen Sie dann vielleicht noch Werte speichern, die für diese Funktion benötigt wird. In den meisten dieser Fälle ist ein Modul dafür am besten geeignet.

Konstanten können zwei spezielle Zwecke haben, wenn man Sie Modulen hinzufügt. Sie können verwendet werden, um die Interface-Elemente einer Applikation zu lokalisieren und um Windows-Shortcuts anzubieten.

Inhalt

- Was sind Module?
- Einfügen von Methoden in ein Modul
- Einfügen von Konstanten in ein Modul
- Einfügen von Eigenschaften in ein Modul

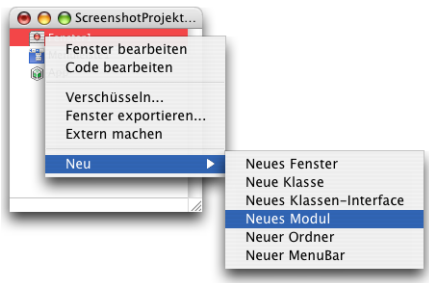
Was sind Module?

In der objektorientierten Umgebung von REALbasic sind geschützte Methoden, Konstanten und Eigenschaften normalerweise Teile von Objekten. Sie sind nur über diese Objekte erreichbar.

Module sind keine Objekte und besitzen auch keine übergeordnete Klasse. Sie werden nie mit dem New-Befehl als Instanz im Speicher angelegt. Nachdem Sie ein Modul in Ihrem Projekt angelegt haben, sind seine Methoden, Konstanten und Eigenschaften sofort verfügbar. Die einzige Ausnahme bilden private Methoden, Eigenschaften oder Konstanten, die zu diesem Modul gehören. Als privat deklarierte Teile eines Moduls, können nur innerhalb des Moduls angesprochen werden.

Anlegen eines neuen Moduls

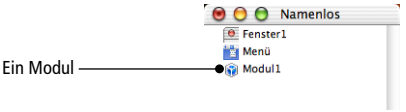
Ein neues Modul erzeugen Sie, indem Sie den Menüeintrag **Ablage/Neues Modul** auswählen. Sie können aber auch das Kontextmenü im Projektfenster verwenden und dort den Menüpunkt **Neu/Neues Modul** wählen.



Das neue Modul erscheint in Ihrem Projektfenster und erhält einen vorgegebenen Namen (das erste heißt „Module1“). Sie können ihm einen eigenen Namen geben, indem Sie diesen im Eigenschaftenfenster eingeben. Für finanzmathematische Funktionen bietet sich zum Beispiel „Financial“ an.

Module können nur Methoden, Konstanten und Eigenschaften enthalten. Diese können nur im Code-Editor verändert werden. Um den Code-Editor zu einem Modul zu öffnen, müssen Sie einen Doppelklick auf dem Modul im Projektfenster ausführen. Module sind an ihrem Piktogramm in diesem Fenster sofort zu erkennen.

Abb. 192: Ein Modul im Projektfenster



Gültigkeitsbereich von Methoden, Eigenschaften und Konstanten eines Moduls

Global: Globale Methoden, Eigenschaften oder Konstanten können innerhalb des gesamten Projekts verwendet werden. Der globale Gültigkeitsbereich ist nur für Elemente eines Moduls verfügbar. Sie können beispielsweise eine globale Eigenschaft als eine Information betrachten, die in allen Teilen Ihres Projekts verfügbar ist. Sie greifen auf eine globale Methode, Eigenschaft oder Konstante über ihren Namen zu.

Öffentlich: Eine öffentliche Konstante, Eigenschaft oder Methode ist ebenfalls in allen Teilen eines Projekts verfügbar. Allerdings müssen Sie über Punkt-Notation darauf zugreifen. Wenn Sie eine öffentliche Eigenschaft „meineEigenschaft" im Modul „Modul1" erzeugen, können Sie diese über „Modul1.meineEigenschaft" erreichen. Innerhalb von Modul1 können Sie die Eigenschaft einfach über die Bezeichnung verwenden.

Privat: Private Konstanten, Methoden oder Eigenschaften sind nur innerhalb des Moduls verfügbar, in dem sie deklariert wurden. Alle anderen Teile eines Projekts können nicht darauf zugreifen.

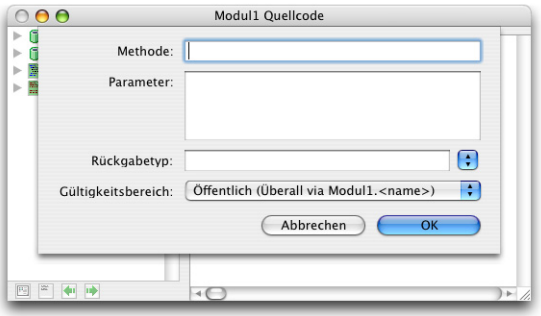
Einfügen von Methoden in ein Modul

Methoden werden in ein neues Modul genauso eingefügt, wie Sie es bei einem Fenster machen würden:

Zum Einfügen einer Methode:

1. Wenn der Code-Editor für das Modul nicht bereits geöffnet ist, führen Sie einen Doppelklick auf das Modul im Projektfenster aus, um ihn zu öffnen oder drücken Sie alt-Tab.
2. Wählen Sie **Bearbeiten/Neue Methode**. Die Dialogbox zur Deklaration einer Methode erscheint.

Abb. 193: Das Sheet-Fenster für die Methoden-Deklaration



3. Geben Sie den Namen und die Parameter der Methode ein.

Wenn Sie Parameter angeben, bestimmen Sie den Datentyp jedes einzelnen Parameters. Wenn Sie beispielsweise eine Fließkommazahl übergeben wollen, die innerhalb der Methode über die Bezeichnung „X“ erreichbar ist, so geben Sie folgenden Parameter an:

```
X As Double
```

Wenn Sie mehrere Parameter angeben wollen, trennen Sie diese mit einem Komma. Um ein Array zu übergeben, fügen Sie der Bezeichnung leere Klammern an. In unserem Beispiel von eben wäre das:

```
X() As Double
```

4. Soll die Methode eine Funktion sein, wählen Sie einen Datentyp für den Rückgabewert. Sie können Arrays oder einzelne Werte zurückgeben. Für ein Array fügen Sie leere Klammern an den Datentyp. Für ein Integer-Array wäre dies:
   ```
   Integer()
   ```
5. Wählen Sie den Gültigkeitsbereich der Methode.
6. Klicken Sie auf **OK**.

   Der Code-Editor für das Modul erscheint und zeigt die neu angelegte Methode an. Wenn der Gültigkeitsbereich Privat ist, heißt der Sub-Ausdruck „Private Sub…“, andernfalls nur „Sub…“.

Klassenerweiterungsmethoden

Eine Klassenerweiterungsmethode wird mit einer Syntax aufgerufen, die bereits andeutet, dass die Methode zu einem anderen Objekt gehört. Zum Beispiel können Sie eine Methode schreiben, die von einem FolderItem-Objekt aufgerufen werden kann.

Um eine Methode als Klassenerweiterungsmethode zu definieren, benutzen Sie das Schlüsselwort „Extends“ vor dem ersten Parameter. Der Datentyp des ersten Parameters ist der Datentyp der Objekte, aus denen die Methode aufgerufen werden darf. Ein Beispiel:

```
Sub meineMethode(Extends f As FolderItem, Name As String)
```

Wird aufgerufen mit:

```
Dim f As FolderItem
f.meineMethode(“Test”)
```

Abb. 194: Deklarieren einer Klassenerweiterungsmethode für die FolderItem-Klasse.

Das Extends-Schlüsselwort kann nur für Methoden eines Moduls verwendet werden.

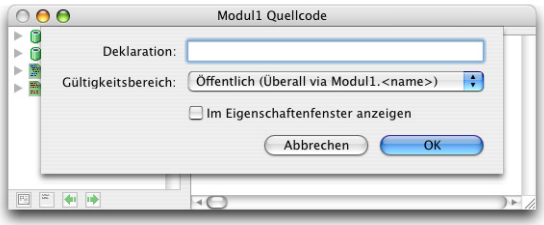
Einfügen von Eigenschaften in ein Modul

Modul-Eigenschaften werden wie Fenster-Eigenschaften zu einem Projekt hinzugefügt. Bestimmen Sie den Gültigkeitsbereich der Eigenschaft, um festzulegen, welche Teile Ihres Projekts darauf zugreifen können.

Zum Einfügen einer Eigenschaft:

1. Öffnen Sie das Modul mit einem Doppelklick im Projektfenster. Der Code-Editor für dieses Modul erscheint.
2. Wählen Sie **Bearbeiten/Neue Eigenschaft**.

Abb. 195: Die Dialogbox für die Eigenschaft-Deklaration



3. Geben Sie der Eigenschaft einen Namen und einen Datentyp. Wollen Sie beispielsweise einen Namen speichern, dann geben Sie **Name As String** ein.

   Die Eigenschaft kann auch ein Array sein. Möchten Sie ein String-Array mit vier Elementen deklarieren, das Vorname, Nachname, Adresse und Telefonnummer speichert, so schreiben Sie:

   aNamen(3) As String

   Sie müssen die Größe des Arrays angeben, können diese aber später im Code mit der Redim-Methode verändern.
4. Wählen Sie einen Gültigkeitsbereich aus und lassen Sie sich die Eigenschaft – falls gewünscht – im Eigenschaftenfenster anzeigen.

   Ist eine Eigenschaft privat, wird ihr Icon im Code-Editor mit einem Warndreieck angezeigt.
5. Klicken Sie **OK**, um die Eigenschaft zu speichern.

   Wenn Sie **OK** klicken, erscheint die Eigenschaft in der Gruppe der Eigenschaften. Der Code-Editor zeigt die Deklaration der neuen Eigenschaft an.
6. Jetzt können Sie noch Notizen und Kommentare zu der Eigenschaft eingeben.

Wenn Sie ein Modul nur anlegen wollen, um globale Eigenschaften unterzubringen, sollten Sie in Erwägung ziehen, diese statt dessen in der App-Klasse zu deklarieren. Die App-Klasse wird automatisch zusammen mit Ihrem Projekt angelegt. Sie haben damit auch globale Eigenschaften zur Verfügung, die aber direkt mit dem Projekt verbunden sind und nicht in einem Modul ausgegliedert sind, das zufällig eben Bestandteil des Projektes ist. Mehr Informationen dazu finden Sie im Abschnitt „Die Application-Klasse“ auf Seite 320.

Einfügen von Konstanten in ein Modul

Globale Konstanten können keine nicht druckbaren Zeichen wie Return, Tab, Space usw. enthalten. Eine Möglichkeit, ein globales Objekt zu erzeugen, das ein solches Zeichen enthält, besteht darin, einem Modul eine Funktion hinzuzufügen, die das gewünschte Zeichen ausgibt. Um z.B. ein Objekt zu erzeugen, dass ein Carriage Return-Zeichen (ASCII 13) ausgibt, fügen Sie Ihrem Modul folgende Funktion hinzu:

```
Function CR as String
Return Chr(13)
End Function
```

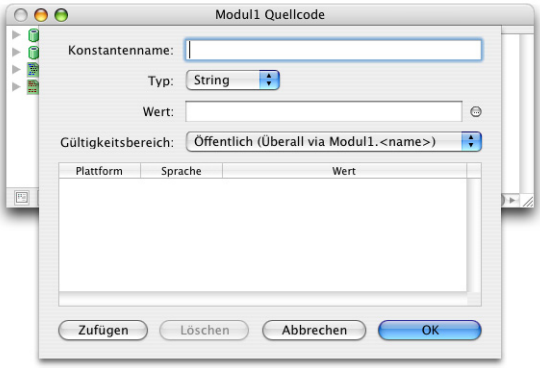
Eine Konstante einem Modul hinzufügen

Um in einem Modul eine Konstante anzulegen, gehen Sie folgendermaßen vor:

1. Klicken Sie im Projektfenster doppelt auf das Modul, um es im Code-Editor zu öffnen.

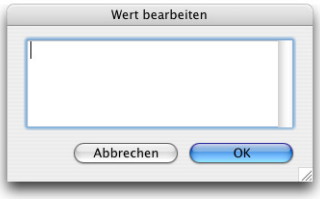
2. Rufen Sie **Bearbeiten/Neue Konstante** auf.

Abbildung 196. Der Dialog „Neue Konstante“:



3. Geben Sie der Konstanten einen Namen, legen sie ihren Datentyp, Gültigkeitsbereich und ihren Wert fest.

   Möchten Sie einen mehrzeiligen Text eingeben, klicken Sie auf den kleinen Button, neben dem Wert-Editfield. Daraufhin erscheint folgender Dialog, in dem Sie mehrzeiligen Text eingeben können.

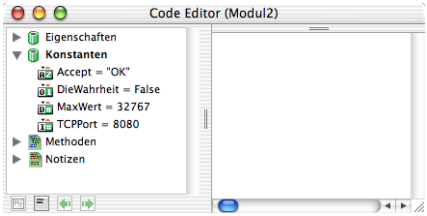


4. Klicken Sie auf **OK**.

Die Dialogbox „Neue Konstante“ unterstützt Standard-Cut/Copy/Paste-Operationen.

Wenn Sie die „Neue Konstante“-Dialogbox bestätigen, erscheint die Konstante in der Objektliste des Moduls. Der Wert der Konstanten wird neben einem Icon, das auf seinen Datentyp hinweist, angezeigt. Folgende Abbildung zeigt den Code-Editor eines Moduls, nachdem mehrere Konstanten hinzugefügt wurden.

Abb. 197: Der Code-Editor eines Moduls mit Konstanten aller möglichen Datentypen

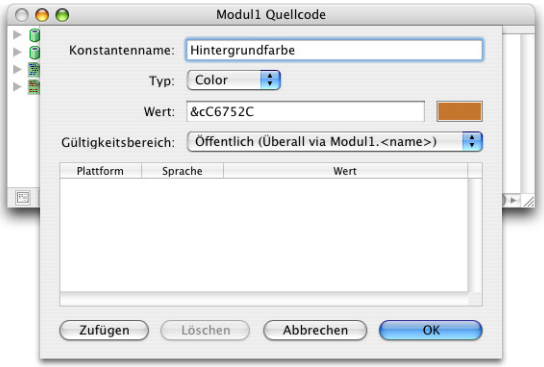


Farbkonstanten

Sie können Konstanten vom Typ Color erzeugen. Farbkonstanten stellen sicher, dass Ihre Applikation einheitliche Farben verwendet. Wenn Sie eine Farbkonstante erzeugen, bestimmen Sie deren RGB-Wert (Rot, Grün, Blau). Sie legen den Wert jeder Komponente mit folgenden Format fest:

&c*RRGGBB*

Dabei ist RR der Rotanteil, GG der Grünanteil und BB der Blauanteil. Der „Neue Konstante“-Dialog macht die Auswahl der Farbe sehr einfach, da er einen Colorpicker enthält, der erscheint, wenn Sie den Typ Color auswählen. Wenn Sie mit dem Colorpicker eine Farbe auswählen, wird diese beim Schließen automatisch übernommen und als Wert eingetragen.



Lokalisieren mit Konstanten

Globale Konstanten bieten einen bequemen Weg, eine Applikation zu lokalisieren. Wenn Sie globale Konstanten für die Texte in Ihrer Applikation verwenden, können Sie Ihre Applikation lokalisieren, indem Sie die Standardsprache in den Projekteinstellungen ändern und die Sprache in den Compiler-Einstellungen, bevor Sie die Applikation compilieren. Weitere Informationen finden Sie im Abschnitt „Das Programm erzeugen“ auf Seite 388.

Der Lokalisierungstabelle im unteren Bereich der Dialogbox „Neue Konstante“, erlaubt es, der Konstanten abhängig von der Plattform und der Sprache verschiedene Werte zuzuweisen. Wenn Sie im Projekt-Einstellungen- oder im Applikation erzeugen-Dialog die Sprache ändern, werden für die verschiedenen Konstanten automatisch die entsprechenden Werte eingesetzt. Sie müssen nur Werte für die verschiedenen Sprach- und Plattform-Kombinationen bestimmen.

Sie können unter folgenden Plattformen wählen:

- Windows
- Macintosh: jedes Mac OS, auf dem REALbasic lauffähig ist
- Linux
- Macintosh (Classic): jede Mac OS Classic Version, auf der REALbasic lauffähig ist
- Macintosh (Carbon): PEF Format; läuft unter klassischen Mac OS mit CarbonLib und unter Mac OS X
- Macintosh: Mach-O-Format, läuft nur unter Mac OS X

Diese Auswahl erlaubt es Ihnen, für die verschiedenen Betriebssysteme unterschiedliche Definitionen Ihrer Konstanten vorzunehmen. Im Folgenden wird gezeigt, wie man die Beschriftung eines Buttons lokalisiert, der zum Abbrechen einer Aktion verwendet wird.

Um eine Konstante zu vereinbaren, die den Caption-Text eines Buttons enthält, gehen Sie folgendermaßen vor:

1. Rufen Sie **Bearbeiten/Neue Konstante** auf und legen Sie eine neue Konstante an.
2. Geben Sie einen Wert für die Standardsprache ein und setzen Sie den Typ auf String.
3. Wählen Sie einen Gültigkeitsbereich für die Konstante

   Üblicherweise sind Konstanten für die Lokalisierung global verfügbar, wenn Sie in Menüs oder Steuerelementen verwendet werden.
4. Klicken Sie auf **Hinzufügen** im unteren Bereich.

   REALbasic fügt eine leere Zeile in die Tabelle ein, wie in folgender Abbildung gezeigt.

Abb. 198: Lokalisieren des Abbrechen-Knopfes für die deutsche Version

Die Plattformspalte legt die Betriebssystemplattform fest, für die die Konstante verwendet werden soll. In der Sprachspalte können Sie neben der aktuellen Systemsprache jede beliebige Sprache auswählen.

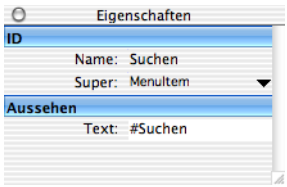
5. Wählen Sie die gewünschte Kombination aus Plattform und Sprache und bestimmen Sie einen Wert.
6. Wiederholen Sie Schritt 5 für jede weitere Kombination, die Sie benötigen.
7. Speichern Sie die Konstante, indem Sie **auf** OK klicken.

Es ist zwingend nötig, dass für jede Plattform/Sprache-Kombination, die Sie anbieten wollen, ein Wert gesetzt wird.

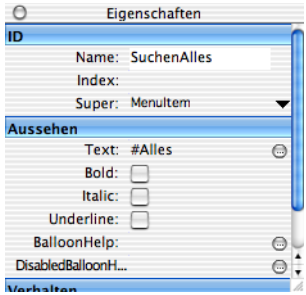
Auch Menüs und Menüeinträge können so lokalisiert werden. Vereinbaren Sie für jeden Text, der als Menü oder Menüeintrag verwendet wird, eine globale Konstante. In die Text-Eigenschaft des Menüs tragen Sie dann den Namen der entsprechenden Konstanten mit vorangestelltem Nummernzeichen („**#**“) ein.

Die folgende Abbildung zeigt ein lokalisiertes Menü.

Abb. 199: Lokalisieren eines Menüs und eines Menüpunkts



Menü



Menüpunkt

Diese Technik kann für alle statischen Texte verwendet werden, die in Fenstern erscheinen: BevelButton-Menüs, Kontextmenüs, Karteireiterbeschriftungen etc. Sie können auf die Konstante verweisen, indem Sie ihren Namen im Eigenschaftenfenster eingeben und ein Nummerzeichen (#) davor setzen.

Sie können Konstanten auch in den Eingabefeldern der Compiler-Einstellungen verwenden, in die Sie statischen Text eingeben können.

REALglot zum Lokalisieren der Applikation einsetzen

Auf der REALbasic CD befindet sich ein Dienstprogramm namens REALglot, mit dem es sehr komfortabel möglich ist, alle nötigen Konstanten für eine Lokalisierung hinzuzufügen.

REALglot durchsucht Ihr Programm und überprüft, ob Sie Konstanten für alle Interface-Elemente verwendet haben und erlaubt Ihnen dann, diese Konstanten zu bearbeiten.

REALglot verwendet die Ausdrücke *Source Language* für die Ausgangssprache und *Target Language* (Zielsprache) für die Sprache, in die das Programm übersetzt werden wird. Die Ausgangssprache ist normalerweise die Sprache, für die Sie Ihr Programm schreiben. Das ist die Sprache, die Sie für die Beschriftung der Knöpfe, der Menüeinträge usw. verwenden. Derjenige, der Ihr Programm lokalisiert, wird diese Sprache als Referenz verwenden, da alle Ihre Konstanten in der Ausgangssprache Werte haben werden.

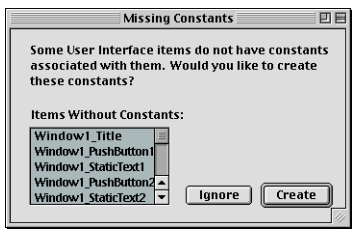
Die Vorgehensweise ist folgende:

- Suchen Sie mit REALglot alle Interface-Elemente, die keine Konstanten verwenden.
- Fügen Sie dem Programm, wo nötig, Konstanten hinzu.
- Erzeugen Sie für die Ausgangssprache Instanzen aller Konstanten.
- Wählen Sie die Zielsprache(n) aus und fügen Sie Instanzen jeder Konstante für jede Zielsprache hinzu.
- Exportieren Sie eine Liste der Interface-Elemente als Tabulator-getrennte Text-Datei für jede Zielsprache. Diese Liste enthält eine Zeile für jedes Interface-Element mit Spalten für Plattform, Ausgangssprache und Zielsprache. Die Spalte für die Ausgangssprache ist ausgefüllt, die Spalte für die Zielsprache ist jedoch leer. Diese Datei senden Sie an den Übersetzer. Der Übersetzer fügt die korrekten Ausdrücke in die Spalte für die Zielsprache ein und schickt Ihnen die vervollständigte Datei zurück.
- Sie importieren dann die vervollständigte Textdatei in REALglot und gleichen sie mit Ihrem Projekt ab. Wenn Sie es danach in REALbasic öffnen, ist das Projekt für die Zielsprache lokalisiert.

  REALglot lädt nur Projekte, die im XML-Format gespeichert sind. Es gibt jedoch eine Voreinstellung, bei der Sie ein REALbasic-Programm auswählen können, das dazu verwendet wird, das Projekt ins XML-Format zu konvertieren. Wenn diese Voreinstellung selektiert ist, können Sie ein beliebiges REALbasic-Projekt öffnen und REALbasic speichert dieses für Sie zuerst im XML-Format.

So lokalisieren Sie Ihr Projekt mit REALglot:

1. Speichern Sie eine Kopie Ihres REALbasic-Projektes im XML-Format. Dies machen Sie, indem Sie „Ablage/Sichern als“ anwählen, im „Format“-Popup „XML“ auswählen und dann auf „Sichern“ klicken.
2. Starten Sie REALglot und öffnen Sie die eben erzeugt XML-Datei.
   Wenn Sie die XML-Datei das erste mal öffnen, durchsucht REALglot Ihr Projekt nach doppelten Konstanten. Wenn solche gefunden werden, erhalten Sie einen Dialog, der eine Liste dieser Elemente enthält. Sie sollten Konstanten, die einen doppelten Namen enthalten, umbenennen, damit Sie sicher sein können, dass Ihr Programm sich wie gewünscht verhält.
   Danach öffnet sich eine Dialogbox, die alle Interface-Objekte auflistet, denen keine Konstanten zugewiesen wurden. Dies zeigt folgende Abbildung:



Sie sollten für alle Interface-Elemente Konstanten verwenden, damit Sie die Applikation komplett lokalisieren können.

3. Falls nötig, wählen Sie die Interface-Elemente in der Liste aus, für die Sie Konstanten erzeugen möchten und klicken dann auf **Create**.
   Ihr XML-Projekt wird in einem Fenster angezeigt.

   Das Language-Popup-Menü enthält eine Liste aller unterstützten Sprachen. Als erstes sollten Sie Instanzen für die Ausgangssprache erzeugen.

4. Wählen Sie die Ausgangssprache im Language-Popup-Menü aus und markieren Sie alle Interface-Elemente, die keine Konstanten besitzen.
   Nun wird der **Make**-Knopf anwählbar.

5. Klicken Sie auf den **Make**-Knopf, um Konstanten für die Ausgangssprache zu erzeugen.
   Als nächstes müssen Sie Konstanten für alle benötigten Zielsprachen hinzufügen.

6. Selektieren Sie eine Zielsprache aus dem Language-Popup-Menü und markieren Sie alle Interface-Elemente, die noch keine Konstanten besitzen und klicken Sie dann auf **Make**.

7. Wiederholen Sie diesen Schritt für alle benötigten Zielsprachen.
   Wenn Sie damit fertig sind, können Sie die Textdateien exportieren. Sie müssen so viele Textdateien wie Zielsprachen erzeugen.

8. Um die Konstanten und ihre Werte zu exportieren, wählen Sie den Menüpunkt **File/Export Constants** an. Es erscheint die Dialogbox **Export Languages**.

   Sie müssen die Ausgangs- und die Zielsprache auswählen. Die Zielsprache (Target) ist die, für die Sie die Lokalisierung durchführen möchten.

Wählen Sie **Source** und **Target** aus und klicken Sie auf **Export**.
Wenn jemand anderes für Sie die Lokalisierung Ihres Projektes übernimmt, können Sie diesem die exportierte Datei schicken. Diese durch Tabs getrennte Textdatei kann mit einem beliebigen Texteditor oder mit dem in REALglot einge-

bauten Konstanten-Editor bearbeitet werden. So sieht in etwa eine kleine Textdatei aus:

Tabelle 12. Beispiel einer von REALglot erzeugten Textdatei

Konstanten-Name	Plattform	English	German
ButtonCaption	Any	OK	
Window1_Title	Any	Untitled	
Menu_File	Any	File	
Menu_FileQuit	Any	Quit	
Menu_Edit	Any	Edit	
Menu_EditUndo	Any	Undo	
Menu_EditCut	Any	Cut	
Menu_EditCopy	Any	Copy	
Menu_EditPaste	Any	Paste	
Menu_EditClear	Any	Clear	

Weitere Informationen finden Sie auf der REALbasic-CD. Die exportierte Datei enthält eine Spalte für den Namen der Konstanten, eine Spalte für die Plattform, eine Spalte für die Ausgangssprache und eine Spalte für die Zielsprache. Derjenige, der die Applikation lokalisiert, füllt die Spalte mit der Zielsprache für alle Einträge aus, die leer sind und schickt Ihnen danach die Datei zurück.
Nachdem die Lokalisierung der Konstanten erfolgt ist, wird die Datei in Ihr XML-Projekt mittels REALglot importiert.

9. Um die Datei mit den lokalisierten Konstanten zu importieren, öffnen Sie zuerst die XML-Version Ihres Projektes in REALglot.
10. Wählen Sie den Menüpunkt **File/Import Constants** aus, um die ausgefüllte Textdatei zu importieren.
    REALglot fügt die Werte der Zielsprache aus der Datei in die XML-Kopie Ihres Projektes ein.
11. Speichern Sie Ihr aktualisiertes Projekt mit **File/Save**.
12. Öffnen Sie in REALbasic die XML-Datei.
    Sie werden bemerken, dass Sie nun lokalisierte Konstanten für die importierte Sprache haben.
    REALglot unterstützt derzeit nur Interface-Elemente. Wenn Sie in Ihrem Code Strings haben, die angezeigt werden, müssen Sie für diese selbst in REALbasic Konstanten erzeugen, damit diese für den Export in REALglot zur Verfügung stehen.

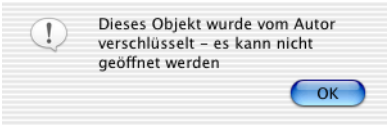
Importieren und exportieren von Modulen

Module können von anderen REALbasic-Projekten importiert werden. Exportierte Module erscheinen mit einem Würfel-Piktogramm auf dem Desktop.

Abb. 200: Ein exportiertes Modul auf dem Desktop

Module können vor dem Exportieren verschlüsselt werden, um zu verhindern, dass andere Entwickler an Ihren Code herankommen. Wenn ein verschlüsseltes Modul in ein anderes Projekt importiert wird, erscheint es im Projektfenster mit einem Schlüssel im Icon( ); wenn der Entwickler darauf einen Doppelklick ausführt, um den Code-Editor aufzurufen, erscheint stattdessen eine Warnbox.

Abb. 201: Warnbox bei verschlüsseltem Modul



Der Entwickler kann das verschlüsselte Modul im Projekt nutzen, er kann jedoch nicht auf den Code des Moduls zugreifen. Verschlüsselung bietet eine gute Möglichkeit, eigenständige Module, die Erweiterungen für andere Programm bieten, zu verkaufen, ohne das Risiko, dass die eigene Arbeit gestohlen wird.

Importieren

Um ein Modul in ein Projekt zu importieren, ziehen Sie das Modul-Piktogramm einfach auf das Projektfenster. Alternativ können Sie das Modul auch über Ablage/Import in der Dateiauswahlbox auswählen. Falls das Modul geschützt ist, erscheint im Projektfenster die verschlossene Version des Modul-Icons.

Wenn Sie ein Modul unter zwei oder mehr Projekten aufteilen möchten, können Sie es als externes Projekt-Element importieren. Weitere Informationen finden Sie im Abschnitt „Externe Projektelemente“ auf Seite 309.

Exportieren

Module können zum Gebrauch in anderen REALbasic-Projekten exportiert werden. Dazu gibt es zwei Möglichkeiten:

- Ziehen Sie das Modul aus dem Projektfenster auf den Schreibtisch.
- Klicken Sie auf das Modul im Projektfenster und wählen Sie **Datei/Modul exportieren**.

Sie können beide Methoden für verschlüsselte Module verwenden.

Wenn Sie den Ordner, in den Sie das Modul exportieren möchten, auf dem Desktop nicht sehen können, ist die erste Methode praktischer. Wenn Sie sich zuerst zu einem bestimmten Ordner vorarbeiten müssen, ist die zweite Methode praktischer. Beide Methoden exportieren das Modul in seinem aktuellen Zustand – verschlüsselt oder unverschlüsselt. Verschlüsselte und unverschlüsselte Module haben das gleiche Desktop-Icon; der Status der Verschlüsselung eines Moduls ist nur im Projektfenster zu erkennen.

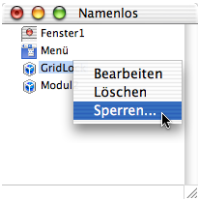
Module schützen

Sie können ein Modul im Projektfenster verschlüsseln (schützen) oder entschlüsseln (den Schutz entfernen). Wenn ein Modul verschlüsselt ist, kann niemand (einschließlich des Autors) auf dessen Code zugreifen, ohne das Passwort anzugeben.

Wenn Sie ein Modul sperren, verwenden Sie hierzu ein Passwort, das später wieder zum Entsperren verwendet werden kann. In diesem Zusammenhang können Sie auch die Checkbox **V3 Verschlüsselung oder höher verw.** aktivieren, wenn Sie die Rückwärtskompatibilität zu früheren REALbasic-Versionen garantieren wollen (in REALbasic 4.5 und höher können Sie das 2.1 Dateiformat nicht mehr verwenden). Wenn Sie sich nicht für diese Option entscheiden, dann wird die Verschlüsselung der Version 4.5 verwendet. Sie sind dann nicht mehr in der Lage, das Modul mit früheren Versionen von REALbasic zu verwenden.

Um ein Modul zu sperren, selektieren Sie dieses im Projektfenster und wählen **Bearbeiten/Sperren** oder wählen Sie nach ctrl-Klick (Windows: Rechtsklick) auf den Modulnamen aus dem Kontextmenü den Menüpunkt **Sperren** aus.

Abb. 202: Das Kontextmenü im Projektfenster



Es erscheint die Dialogbox zum Verschlüsseln des Moduls.

Abb. 203: Der Verschlüsseln-Dialog

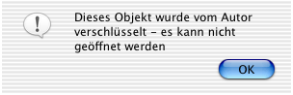
Geben Sie ein Passwort für die Verschlüsselung ein und bestätigen Sie dieses. Selektieren Sie **V3 Verschlüsselung oder höher verw.** nur, wenn das Modul mit der Version 3-4 von REALbasic verwendet wird.

Wenn Sie diese Option nicht aktivieren, wird die Verschlüsselung der Version 4.5 verwendet. Sie können das Modul dann nicht mit REALbasic-Versionen kleiner 4.5 verwenden.

Wichtig: Vergessen Sie nicht ihr Passwort.

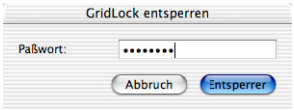
Ein gesperrtes Modul erscheint im Projektfenster mit einem Schlüssel in der rechten unteren Ecke des Modul-Icons  . Wenn ein Anwender versucht, ein gesperrtes Modul im Fenstereditor zu öffnen, erscheint folgende Warnbox:

Abb. 204: Der Dialog „Objekt verschlüsselt“



Um das Modul oder seinen Code zu bearbeiten, müssen Sie es mit dem entschlüsseln. Selektieren Sie es dazu im Projektfenster und rufen Sie den Menüpunkt **Bearbeiten/Entschlüsseln** auf (bzw. Entschlüsseln im Kontextmenü).

Abb. 205: Der Entschlüsseln-Dialog



Geben Sie das Passwort ein und klicken Sie auf **Entsperren**. Nach kurzer Zeit verschwindet der Schlüssel aus dem Icon des Moduls, um anzuzeigen, dass die Entschlüsselung erfolgreich war. Wenn Sie ein falsches Passwort eingegeben haben, informiert Sie eine Dialogbox darüber. Ohne Passwort gibt es keine Möglichkeit, das Modul zu entschlüsseln.

Kapitel 7 Text und Grafik verwenden

Die allermeisten Programme müssen Texte oder Grafiken verarbeiten oder darstellen. Deshalb stellt REALbasic zahlreiche Funktionen zum Erzeugen, Manipulieren, Anzeigen und Drucken von Texten und Grafiken zur Verfügung.

Auf der Basis des Canvas-Steuerelements können eigene Steuerelemente entwickelt werden.

Inhalt

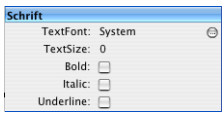
Verwenden von Fonts

Für viele Objekte und Steuerelemente können Sie die Schrift, die Schriftgröße und den Schriftstil frei wählen. EditField-Steuerelemente können mehrere Schriften in verschiedenen Stilen und Größen verwenden. Dies wird als Styled Text bezeichnet (mehr dazu siehe „Styled Text“ auf Seite 244). ListBoxen unterstützen ebenfalls unterschiedliche Schriftstile. Steuerelemente, die eine einzelne Schrift verwenden, verfügen über eine TextFont-Eigenschaft, mit der man diese Schrift festlegen kann.

Der Systemfont und der kleine Systemfont

Der Systemfont ist die Schriftart, die vom Betriebssystem als Standardfont eingesetzt wird. Er wird auch für die Menüs verwendet.

Um Texte im Systemfont auszugeben, können Sie der TextFont-Eigenschaft des betreffenden Objekts im Eigenschaftenfenster den Wert „System“ zuweisen. Wenn Sie zusätzlich 0 (Null) als Schriftgröße einstellen, wird REALbasic automatisch die Schriftgröße verwenden, die optimal für die Betriebssystemplattform geeignet ist, auf der Ihr Programm gerade läuft. Da z.B. Windows und Mac OS unterschiedliche dpi-Werte für die Bildschirmauflösung annehmen, werden auch unterschiedliche Schriftgrößen benötigt. Wenn Sie die Schriftgröße 0 wählen, verwendet Ihr Programm automatisch plattformabhängig die passende Schriftgröße, ohne dass Sie für jede Plattform eigene Fenster kreieren müssen.



Falls das verwendete Betriebssystem einen großen und einen kleinen Systemfont unterstützt, können Sie als TextFont auch „SmallSystem“ einstellen. Damit wird der kleine Systemfont aktiviert. Ist der kleine Systemfont nicht verfügbar, wird statt dessen der Systemfont verwendet.

Abb. 206: Systemfont und kleiner Systemfont unter Mac OS X

Welche Fonts sind verfügbar?

Wenn Sie nicht nur den Systemfont verwenden wollen, stellt sich die Frage, welche Fonts auf dem Computer des Benutzers überhaupt verfügbar sind. Mit den globalen Funktionen FontCount und Font können Sie auf einfache Weise feststellen, ob ein Font vorhanden ist. Wenn Sie der folgenden Funktion einen Fontnamen übergeben, liefert Sie True oder False zurück, je nachdem, ob der Font gefunden wurde oder nicht:

```
Function FontAvailable(FontName as String) As Boolean
Dim i, nFonts as Integer
nFonts=FontCount-1
For i=0 to nFonts
 If Font(i)=FontName Then
 Return True
 End If
Next
Return False
End Function
```

Mit folgender kurzen Schleife kann ein PopupMenu oder eine ListBox mit allen verfügbaren Fonts aufgebaut werden:

```
Dim i, nFonts as Integer
nFonts=FontCount-1
For i=0 to nFonts
 Me.AddRow Font(i)
Next
```

Wenn Sie in Ihrem Programm ein Schriftmenü benötigen, können Sie dies folgendermaßen realisieren:

1. Legen Sie ein Menü mit dem Namen „Schrift“ an und setzen Sie seine Text-Eigenschaft ebenfalls auf „Schrift“.
2. Erzeugen Sie im Schrift-Menü einen neuen Menüpunkt und tragen in dessen Text-Eigenschaft **FontName** ein. REALbasic wird den neuen Menüpunkt automatisch **SchriftFontName** nennen.
3. Setzen Sie die Index-Eigenschaft des Menüpunkts auf 0 (null)
4. Legen Sie das folgende Programmstück in den Open-Event-Handler der App-Klasse:

```
Dim m as MenuItem
Dim i, nFonts as Integer
nFonts=FontCount-1
SchriftFontName(0).text=Font(0)
For i=1 to nFonts
 m=New SchriftFontName
 m.text=font(i)
next
```

Alle so erzeugten Menüeinträge können sich einen Menu-Handler teilen. Diesem wird ein Index übergeben, der darüber Auskunft gibt, welcher Menüpunkt ausgewählt wurde. Zusammen mit der Font-Funktion kann dieser Index dazu verwendet werden festzustellen, welche Schrift ausgewählt wurde.

Im REALbasic-Tutorial programmieren Sie ein Font-Menü, das ein BevelButton-Steuerelement an Stelle eines Menüs in der Menüleiste verwendet. Das Font-Menü wird mit der AddRow-Methode des BevelButton-Steuerelements erzeugt.

```
Dim i, nFonts as Integer
nFonts=FontCount-1
For i=0 to nFonts
  me.AddRow Font(i)
Next
me.Caption=TextField.SelTextFont
```

Die letzte Zeile der Methode weist den BevelButton an, den Font, der gerade im Texteditor-Programm verwendet wird, anzuzeigen.

Sie können die gleiche Logik verwenden, um ein Font-Menü mit einem PopupMenu-Steuerelement zu erzeugen, da auch dieses eine AddRow-Methode besitzt.

Verarbeiten von selektiertem Text

Der Begriff „selektierter Text" bezieht sich auf Textteile, die in einem EditField selektiert (also grafisch – meist durch Invertieren – hervorgehoben) sind, das gerade den Fokus besitzt. EditFields haben drei Eigenschaften, die dazu dienen, selektierten Text auszulesen oder Text zu selektieren.

Tabelle 13. Eigenschaften, um selektierten Text zu erhalten oder zu produzieren

Name	Beschreibung
SelLength	Die Anzahl der gegenwärtig ausgewählten Zeichen. Wenn Sie diese Zahl ändern, verändert sich die Selektion. Setzen Sie den Wert auf 0 (null), wird die Selektion aufgehoben und der Cursor landet auf der Position, die in SelStart angegeben ist.
SelStart	Die Zahl der nicht selektierten Zeichen vor dem selektierten Text. Selektiert man beispielsweise das fünfte Zeichen eines Textes, steht in SelStart eine Vier. Setzt man den Wert auf Null, beginnt die Selektion am Anfang des EditFields.
SelText	Eine Zeichenkette, die den gesamten selektierten Text enthält. Wird hier ein anderer Inhalt abgelegt, wird der selektierte Text durch diesen ersetzt. Ist kein Text selektiert, dann wird der in SelText geschriebene String an der Einfügemarke (Wert von SelStart) ausgegeben.

Folgende Zeilen wählen den gesamten Text in dem EditField aus, das gerade den Fokus hat:

```
EditField1.SelStart=0
EditField1.SelLength=Len(EditField1.Text)
```

Soll ein bestimmter Programmcode ausgeführt werden, wenn der Benutzer den Cursor bewegt oder einige Zeichen selektiert, dann stecken Sie den Code dazu in den SelChange-Event-Handler des EditFields.

Anlegen eines Passwort-Feldes

EditFields haben die Eigenschaften Password und LimitText, die man für Passwort-Felder verwenden kann. Ist die Eigenschaft Password gesetzt, dann werden Buchstaben als Punkte (•) ausgegeben. Die korrekten Buchstaben landen aber dennoch in der Text-Eigenschaft des EditFields. Mit der Eigenschaft LimitText kann man die Maximallänge eines Textes im EditField vorgeben.

Achtung: Die Password-Eigenschaft funktioniert nur dann, wenn die Eigenschaft MultiLine nicht aktiviert, also False ist.

Textfeldeingaben formatieren und filtern

Das EditField besitzt zwei Eigenschaften, die es Ihnen erlauben, den eingegebenen Text Zeichen für Zeichen zu formatieren und zu filtern. Dies kann zum Beispiel für eine Datenbank-Eingabemaske verwendet werden, in der die Kundennummer das Format „12-123-1234“ erfüllen muss.

Die Format-Eigenschaft

Die Format-Eigenschaft eines EditFields erlaubt es, den eingegebenen Wert anders darzustellen, als er tatsächlich eingegeben wurde. So können Sie beispielsweise Zahlen immer mit zwei Stellen hinter dem Komma darstellen. Wenn das EditField den Fokus erhält, wird die Eingabe ohne Formatierung angezeigt. Ohne Fokus wird der Wert mit entsprechender Formatierung angezeigt. Die unterstützten Formate entsprechen denen der Format-Funktion. Lesen Sie dazu den Eintrag über die Format-Funktion in der Sprachreferenz. Um die Formatierung zu löschen, geben Sie zwei Anführungszeichen an (einen leeren String). Hier einige Beispiele:

```
// Zahlen immer mit zwei Dezimalstellen anzeigen, notfalls mit Nullen auffüllen.
Editfield1.format = "0.00"

// Zahlen maximal mit zwei Dezimalstellen anzeigen, aber nicht mit Nullen auffüllen.
Editfield1.format = "#.##"

// Die Formatierung aufheben
Editfield1.format = ""
```

Die Mask-Eigenschaft

Mit Hilfe der Mask-Eigenschaft, filtern Sie die Eingabe und können diese somit auf bestimmte Zeichen beschränken. So können Sie die Eingabe der oben erwähnten Kundennummer überprüfen, die ein bestimmtes Format aufweisen muss. Jedes Zeichen in der Maske steht als Platzhalter für einen bestimmten Typ von Zeichen oder ein Zeichen selbst.

Die Eingabemaske ist vollständig mit Visual Basic kompatibel, mit einer Ausnahme: „~“ ist in REALbasic für zukünftige Nutzung oder Erweiterungen reserviert.

Besitzt ein EditField eine Eingabemaske, wird dies dem Benutzer nicht kenntlich gemacht, bis dieser nicht erlaubte Eingaben macht.

Lesen Sie auch den Eintrag zum EditField in der Sprachreferenz.

Hier ein paar Beispiele:

```
// Erlaubt eine Kundennummer, wie weiter oben beschrieben. (Nur Ziffern)
Editfield1.Mask = "##-###-####"
// Erlaubt ein Datum in der Form 14-Dec-1972
Editfield1.Mask = "##-???-####"
// Automatische Großschrift einer Registriernummer der Form AWS-1925-ASD
Editfield1.Mask = ">???-####-???"
// Eingabemaske entfernen
Editfield1.Mask = ""
```

Styled Text

Styled Text bedeutet, dass der betreffende Text nicht nur aus einem Font in einer Größe und mit einem Schriftstil bestehen muss. Um Styled Text in einem EditField verwenden zu können, müssen die Eigenschaften MultiLine und Styled eingeschaltet (True) sein. Um Styled Text auszudrucken, müssen Sie die Klasse StyledTextPrinter verwenden (siehe Seite 274).

Ermitteln von Font, Größe und Stil des Textes.

Die EditFields haben Eigenschaften, die es leicht ermöglichen, den Font, seine Größe und seinen Stil zu ermitteln. Die Eigenschaft SelTextFont kann verwendet werden, um den Font der gerade aktiven Selektion zu bestimmen. Besteht diese nur aus einem Font, enthält SelTextFont den Namen dieses Fonts. Sind es mehrere Fonts, so ist SelTextFont leer.

Die folgende Funktion ermittelt die in einer Selektion verwendeten Fonts. Dazu wird gegebenenfalls in einer Schleife für jedes Zeichen in der Selektion der Font bestimmt:

```
Function Fonts(item as EditField) as String
 Dim fonts, theFont as String
 Dim i, Start, Length as Integer
 If Field.SelTextFont="" Then
 Start=Field.SelStart
 Length=Field.SelLength
 For i=Start to Start+Length
 Field.SelStart=i
 Field.SelLength=1
 If InStr(fonts,Field.SelTextFont)=0 Then
 If fonts="" Then
 fonts=Field.SelTextFont
 Else
 fonts=fonts+", "+Field.SelTextFont
 End if
 End if
 Next
 Return fonts
 Else
 Return Field.SelTextFont
 End If
End Function
```

Die Eigenschaft SelTextSize wird dazu verwendet, die Font-Größe des selektierten Textes zu speichern und wird analog zu SelTextFont benutzt. Haben alle Zeichen der Selektion die gleiche Größe, steht diese in SelTextSize. Ist dies nicht der Fall, so steht dort eine 0.

Es gibt außerdem boole'sche Eigenschaften, mit denen man ermitteln kann, ob alle Zeichen eines selektierten Textes einem bestimmten Schriftstil entsprechen. Da in einem Text mehrere Stilvarianten gleichzeitig verwendet werden können, bestimmen diese Eigenschaften lediglich, ob allen Zeichen des selektierten Textes ein bestimmter Schriftstil zugewiesen wurde.

Wenn beispielsweise alle Zeichen des selektierten Textes den Schriftstil fett (bold) besitzen und einige Zeichen zusätzlich kursiv (italic) sind, liefert ein Test auf bold den Wert True zurück. Der Test auf italic liefert False zurück, da nicht alle Zeichen des selektierten Textes den Schriftstil italic besitzen.

Für jede dieser Eigenschaften müssen alle Zeichen überprüft werden, um dann den Wert False oder True zu ermitteln. Zum Testen der Attribute wird im Programm einfach eine Selektion für jeweils genau ein Zeichen gesetzt und dann werden dessen Attribute einzeln abgefragt. Dies ähnelt dem Vorgehen beim Ermitteln des verwendeten Fonts. Die Eigenschaften für die verschiedenen Schriftstile sind folgende:

Tabelle 14. Eigenschaften die eine Schriftart testen

Eigenschaft	Stil
SelBold	Fett (Bold)
Selltalic	Kursiv (Italic)
SelUnderline	Unterstrichen (Underline)

Tabelle 14. Eigenschaften die eine Schriftart testen

Eigenschaft	Stil
SelOutline	Umrandet (Outline)
SelShadow	Schattiert (Shadow)
SelCondense	Schmal (Condensed)
SelExtend	Breit (Extended)

Die Stilarten Outline, Shadow, Condensed und Extend werden nur von Mac OS Classic unterstützt.

So aktiviert man den Menüpunkt **Fett** (bold), wenn der selektierte Text fett ist:

```
StyleBold.Checked=EditField1.SelBold
```

Ist keines der Zeichen in der Selektion fett, dann liefert EditField1.SelBold den Wert False, der dann der Eigenschaft Checked des Menüpunktes StyleBold zugewiesen wird.

Setzen von Schrift, Größe und Stil

Die Eigenschaften, die zur Ermittlung von Schrift, Größe und Stil dienen, werden auch dazu verwendet, diese Attribute zu verändern. Um beispielsweise einer Selektion den Font Helvetica zuzuweisen, schreiben Sie folgendes:

```
Editfield1.SelTextFont="Helvetica"
```

Denken Sie daran, dass das nur dann funktionieren kann, wenn der Font auch im System des Benutzers installiert ist. Sonst passiert bei der Zuweisung gar nichts. Mit der bereits angesprochenen Funktion FontAvailable kann man herausfinden, ob ein bestimmter Font vorhanden ist.

Die Schriftgröße des selektierten Textes kann mittels SelTextSize geändert werden. Eine Änderung auf 12 Punkt erreicht man so:

```
Editfield1.SelTextSize=12
```

Um einen bestimmten Stil zuzuweisen, setzen Sie einfach das zugehörige Attribut auf True. Fett macht man den Text also so:

```
Editfield1.SelBold=True
```

Tabelle 14 auf Seite 245 zeigt alle Stil-Eigenschaften von EditFields, die alle in derselben Art und Weise benutzt werden können.

Die EditFields verfügen auch über Methoden, mit denen die Stile ein- und ausgeschaltet (toggle) werden können. Durch ein Toggle-Kommando werden alle Zeichen, bei denen das entsprechende Stilattribut nicht gesetzt ist, mit diesem Attribut versehen und bei allen Zeichen, bei denen es gesetzt ist, wird es gelöscht. Dazu ruft man die Toggle-Methode für das entsprechende Attribut auf:

```
Editfield1.ToggleSelectionBold
```

Die Toggle-Methoden finden Sie in der folgenden Tabelle:

Tabelle 15. Methoden zur Veränderung des Stils

Methode	Stil
ToggleSelectionBold	Bold/Fett
ToggleSelectionItalic	Italic/Kursiv
ToggleSelectionUnderline	Underline/Unterstrichen
ToggleSelectionOutline	Outline/Umrandet
ToggleSelectionShadow	Shadow/Schattiert

Tabelle 15. Methoden zur Veränderung des Stils

Methode	Stil
ToggleSelectionCondense	Condensed/Schmal
ToggleSelectionExtend	Extended/Breit

Outline, Shadow, Condensed und Extend werden nur unter Mac OS Classic unterstützt.

Arbeiten mit StyledText Objekten

Wenn Sie mit formatiertem Text in einem EditField arbeiten, können Sie die Eigenschaften des EditFields zum Setzen und Auslesen von Stil-Attributen nutzen (siehe oben). REALbasic bietet Ihnen auch die Möglichkeit, formatierten Text unabhängig von einem EditField zu verwenden. Dafür werden die Methoden und Eigenschaften der StyledText-Klasse verwendet. Die Eigenschaft Text enthält den formatierten Text, der von dem StyledText-Objekt verwaltet wird. Dazu gibt es noch sechs weitere Eigenschaften, mit denen Sie die Formatierung des Texts modifizieren oder auslesen können.

Tabelle 16. Eigenschaften zum Setzen und Auslesen der Formatierung

Name	Beschreibung
Bold	Auslesen oder Setzen des Fettschrift-Attributs des markierten Texts.
Font	Auslesen oder Setzen der Schriftart des markierten Texts.
Italic	Auslesen oder Setzen des Kursiv-Attributs des markierten Texts.
Size	Auslesen oder Setzen der Schriftgröße des markierten Texts.
TextColor	Auslesen oder Setzen der Schriftfarbe des markierten Texts.
Underline	Auslesen oder Setzen des Unterstreichen-Attributs des markierten Texts.

Jede Eigenschaft akzeptiert Parameter für die Startposition und die Anzahl der Zeichen. Wenn Sie zum Beispiel die Fettschrift für einen bestimmten Text verwenden möchten, sähe ein Aufruf so aus:

```
Dim st as StyledText
st.Text="Dies ist fett geschrieben"
st.Bold(9,4)=TRUE
```

Damit wird das Wort „fett“ in Fettschrift geschrieben. Eine Folge von Zeichen mit identischen Attributen bildet ein StyleRun-Objekt. Im Beispiel gibt es drei StyleRun-Objekte. Die Zeichen bis zum Wort „fett“, das Wort „fett“ selbst und die restlichen Zeichen. Jedes StyleRun-Objekt ist die Ausprägung einer Zeichenformatierung im Sinne einer Textverarbeitung. Die gesamte Eigenschaft Text besteht aus solchen StyleRun-Objekten.

Die StyledText Klasse besitzt sechs Methoden, um StyleRun-Objekte zu verwalten:

Tabelle 17. Methoden zum verwalten von StyleRun-Objekten

Name	Beschreibung
AppendStyleRun	Fügt ein StyleRun am Ende des Texts an.
InsertStyleRun	Fügt ein StyleRun an der angegebenen Position ein.
RemoveStyleRun	Entfernt das angegebene StyleRun aus dem Text.
StyleRun	Bietet Zugriff auf ein StyleRun. Die StyleRun Klasse besitzt eigene Eigenschaften, die die vorhandene Zeichenformatierung beschreiben.
StyleRunCount	Liefert die Anzahl der StyleRun-Objekte, die zusammen den Text bilden.
StyleRunRange	Bietet Zugriff auf die Start- und Endposition, sowie die Länge des StyleRun-Objekts.

Der Text eines StyledText-Objekts kann aus mehreren Absätzen bestehen. Als Absatz bezeichnet man einen Textbereich, der sich zwischen zwei Zeilenenden (newline) befindet. Ein Absatz kann entweder mit Hilfe der EndOfLine-Funktion

oder mit einem Zeilenende definiert werden. Das Zeilenende ist ein Zeichen, das sich je nach Plattform unterscheidet. Es gilt jeweils das Zeichen der Plattform, auf der das Programm ausgeführt wird. Ein Absatz kann aus mehreren Style-Run-Objekten bestehen. Der Absatz selbst besitzt nur eine Eigenschaft, die Textausrichtung (links, zentriert, rechts).

Es gibt drei Methoden der StyledText Klasse, die sich auf Absätze beziehen:

Tabelle 18. Methoden, die sich auf Absätze beziehen

Name	Beschreibung
Paragraph	Bietet Zugriff auf einen Absatz im Text. Die Paragraph-Klasse bietet eigene Eigenschaften für die Start- und Endposition, sowie die Länge und Ausrichtung eines Absatzes.
ParagraphCount	Liefert die Anzahl der Absätze, aus denen der Text besteht.
SetParagraphAligment	Bestimmt die Ausrichtung des Absatzes (links, zentriert, rechts).

Auch wenn Sie mit einem StyledText-Objekt in Ihrem Code arbeiten können, ohne dies jemals anzuzeigen, steht ein EditField immer mit einem StyledText-Objekt in Zusammenhang. Dieses Objekt können Sie über die Eigenschaft „StyledText“ des EditFields erreichen. Wenn Sie mit dem StyledText-Objekt eines EditFields arbeiten wollen, müssen Sie die Eigenschaften „MultiLine“ und „Styled“ im Eigenschaftenfenster des EditFields aktivieren.

Beispiel: Die Zeile

```
EditField1.StyledText=”Dies ist mein formatierter Text.” + EndOfLine _
+ “Ist das nicht beeindruckend?”
```

Setzt den Inhalt des EditFields und zeigt den Text an. Jetzt können Sie den Text formatieren:

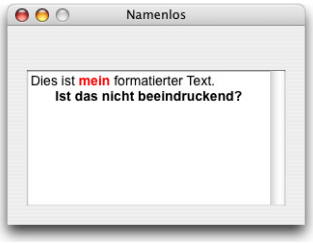
```
Dim st,ln as Integer
Dim Text as String
Text="Dies ist mein formatierter Text."+EndOfLine+"Ist das nicht beeindruckend?”
EditField1.StyledText.Text=Text

//Schriftart und Schriftgröße dem gesamten Text zuweisen
EditField1.StyledText.Font(0,Len(Text))="Arial"
EditField1.StyledText.Size(0,Len(Text))=14

//Die Zeichen des Worts “mein” im ersten Absatz hervorheben.
EditField1.StyledText.Bold(9,4)=True
EditField1.StyledText.textColor(9,4)=&cFF0000 //Rot

//Position des zweiten Absatzes ermitteln (Positionsindex beginnt bei 0).
st=EditField1.StyledText.Paragraph(1).StartPos-1
ln=EditField1.StyledText.Paragraph(1).Length

//Zweiten Absatz fett darstellen
EditField1.StyledText.Bold(st,ln)=True
//Zweiten Absatz zentrieren
EditField1.StyledText.ParagraphAlignment(1)=1
```



In diesem Beispiel wird das StyledText-Objekt des EditFields verwendet, aber Sie können natürlich auch ein eigenes StyledText-Objekt mit Hilfe der Dim-Anweisung erzeugen und dies ohne Bezug auf ein Steuerelement verwenden. Wenn Sie es bearbeitet haben und darstellen wollen, können Sie es der StyledText-Eigenschaft eines EditFields zuweisen.

```
Dim st as StyledText
// Tun Sie, was immer Sie mit dem Text tun wollen
EditField1.StyledText=st
```

Sie können den Text auch als eine Folge von StyleRun-Objekten exportieren und – nachdem Sie ihn wieder importiert haben – mit Hilfe der AppendStyleRun-Methode wieder zusammensetzen. Weitere Informationen über StyleRun- und StyledText-Objekte finden Sie in der Sprachreferenz.

Arbeiten mit Textcodierungen

Jeder Computer verwendet eine bestimmte Codierung, wenn er Zeichenketten als eine Reihe von Bytes speichert. Die älteste und bekannteste Codierung ist wohl ASCII. ASCII definiert Zeichen für den Wertebereich 0-127. Dieser Wertebereich beinhaltet das englische Alphabet, Ziffern, einige Symbole und unsichtbare Steuerzeichen, die in früheren Computern verwendet wurden. Sie können die Chr-Funktion benutzen, wenn Sie das ASCII-Zeichen eines Werts benötigen.

Es wurden einige Erweiterungen zu ASCII eingeführt, die zusätzliche Symbole, Zeichen mit Akzent oder nicht-römische Alphabete codieren. Eine weitere Codierung, Unicode genannt, wurde eigens dafür entwickelt, um sämtliche Sprachen und auch Mixturen der Sprachen in einer Zeichenkette (string) zu ermöglichen. REALbasic bietet zwei Unicode-Codierungen, UTF-8 und UTF-16. Alle Konstanten, Strings und so weiter, werden intern UTF-8 codiert.

Wenn der String in REALbasic erzeugt, gespeichert und wieder geladen wurde, brauchen Sie sich keine Gedanken um die Codierung zu machen, da REALbasic die Codierung des Strings mit dem String speichert. Dies gilt auch für die REALdatabase-Engine.

Wenn Sie allerdings Strings verwenden, die außerhalb von REALbasic erzeugt oder geändert wurden, müssen Sie verstehen, wie die Textcodierung funktioniert und welche Änderungen nötig sind, damit die Verarbeitung der Texte weiterhin reibungslos funktioniert.

Textcodierung: Von ASCII zu Unicode

Wie Sie bereits wissen, speichert der Computer keine Zeichen, sondern nur numerische Werte, die er den Zeichen zuordnet. Zum Beispiel hat ein Zeilenumbruch in ASCII den Wert 13.

Als die Computerindustrie noch in den Kinderschuhe steckte, verwendete jeder Computerhersteller eine eigene Nummerierung, einen sogenannten Zeichensatz (character set). Dieser legte die Zuordnung vom numerischen Wert zu Zeichen (Buchstaben, Ziffern, Steuerzeichen, etc.) fest. Mit einem solchen Zeichensatz (character set) konnte man Informationen zwischen den verschiedenen Computern austauschen.

1963 verabschiedete die American Standards Association (die später in American National Standards Institute umbenannte wurde) den „American Standard Code for Information Interchange“, kurz ASCII, der auf dem Zeichensatz einer englischen Schreibmaschine basierte.

Im Laufe der Zeit wurden Computer auch außerhalb der USA populär und allmählich zeigten sich die Schwächen von ASCII. Die ASCII-Codierung kannte nur 128 Zeichen. Diese umfassten neben den Zeichen einer englischen Schreibmaschine einige Steuerzeichen, um die Ausgabe des Computers zu manipulieren, kannten aber keine Sonderzeichen, die beispielsweise häufig in Büchern verwendet wurden (typografische Anführungszeichen, Apostroph, Punktsymbol für Aufzählungen). Auch viele Sprachen, wie Französisch oder Deutsch, waren nicht Teil von ASCII.

Als der Macintosh und Windows eingeführt wurden, besaß jedes Betriebssystem eine eigene Erweiterung zu ASCII, für den Wertebereich 128-255. Damit waren die Betriebssysteme in der Lage, auch Zeichen mit Akzent oder Umlaute zu

unterstützen. Leider waren diese beiden Erweiterungen nicht kompatibel. Cross-Plattform-Applikationen benötigten deshalb eine eigene Komponente, um mit dem erweiterten Zeichensatz umzugehen.

Das Problem wird allerdings größer, wenn es um Sprachen geht, die kein römisches Alphabet verwenden, wie zum Beispiel Japanisch, Chinesisch oder Hebräisch. Aufgrund der Fülle von Zeichen wurden Zeichensätze erfunden, die zwei Bytes zur Zeichencodierung verwendeten.

Auch Apple erzeugte einige Textcodierungen, um den Umgang mit Daten zu vereinfachen. MacRoman ist die Codierung, die für Dateien mit ASCII-Zeichen verwendet wird. MacJapanese ist für Dateien gedacht, die japanische Zeichen enthalten. Es gibt noch weitere, aber sie alle sind Mac-spezifisch. Der Datenaustausch zwischen den verschiedenen Systemen wurde dadurch nicht einfacher, und auch das Mischen mehrerer Sprachen in einem Dokument blieb problematisch.

Es wurde klar, dass eine umfassende Codierung entwickelt werden musste, die alle vorhandenen Zeichen für alle Sprachen codieren konnte. Diese universelle Codierung wurde von einem Xerox-Mitarbeiter, der an der Entwicklung beteiligt war, „Unicode“ genannt. Unicode schließlich löste all diese Probleme. Jedes beliebige Zeichen würde auf allen Computern gleich sein, sofern diese Unicode unterstützten. Mehr noch, Unicode ermöglicht es, Sprachen in einem Dokument zu mischen, da alle Sprachen in einer Codierung untergebracht waren.

Die Unterstützung von Unicode wurde auf dem Mac mit System 7.6 und unter Windows mit Windows 95 eingeführt. Es war nun möglich, Dateien zwischen den verschiedenen Codierungen zu konvertieren, aber Unicode war immer noch die Ausnahme, nicht die Regel. Erst mit Mac OS X und Windows 2000 änderte sich dies.

Momentan ist man als Computerbenutzer in einer Zwickmühle. Einerseits arbeiten neuere Systeme durchweg mit Unicode, andererseits wird immer noch mit älteren Systemen gearbeitet. Im Ergebnis kann es deshalb vorkommen, dass Sie doch mit unterschiedlichen Codierungen umgehen müssen. Zumindest für eine Übergangszeit. Das bedeutet, dass Sie Ihren Code entsprechend anpassen müssen. In Zukunft wird man annehmen können, dass eine Datei in Unicode codiert ist, aber bis dahin werden Sie Ihre Applikationen so anpassen müssen, dass sie mit unterschiedlichen Codierungen umgehen können.

Anpassen des Codes für unterschiedliche Textcodierungen

Leider gibt es keinen 100% akkuraten Weg, die Codierung einer Datei zu ermitteln. Sie sollten daher wissen, in welcher Codierung die Datei vorliegt. Kommt die Datei beispielsweise von einem englischsprachigen Mac-Benutzer, so wird die Codierung höchstwahrscheinlich MacRoman sein. Bei einem englischsprachigen Windows-Benutzer wohl WindowsANSI. Wenn die Codierung eines Strings definiert ist, können Sie diese folgendermaßen ermitteln:

```
TextEnc=Encoding(s)
```

Bei diesem Beispiel enthält die Variable s den Text und TextEnc ist ein Objekt vom Typ TextEncoding. Ist keine Codierung definiert, wird Nil zurückgegeben.

Lesen von Textdateien

Im folgenden Beispiel wird eine Textdatei eingelesen und in einem EditField angezeigt. Dabei werden keine Annahmen über die Codierung gemacht und der Text wird, falls nicht in Unicode codiert, möglicherweise nicht korrekt dargestellt. Das Beispiel wurde absichtlich kurz gehalten, um nicht vom Thema Codierung abzulenken.

```
Dim f As FolderItem
Dim t as TextInputStream
f = GetFolderItem("Beispiel.txt")
t = f.OpenAsTextFile
Editfield1.text = t.ReadAll
t.close
```

Wenn Sie die Codierung der Datei wissen, können Sie das Beispiel schnell und einfach anpassen, um den Text mit korrekter Codierung einzulesen. Die TextInputStream-Klasse besitzt eine Encoding-Eigenschaft, mit der Sie die Codierung für den einzulesenden Text angeben. Dabei spielt es keine Rolle, ob Sie die Read- oder ReadAll-Methode verwenden.

Im nun geänderten Beispiel wird die Codierung auf MacRoman gesetzt. Die Datei wird nun korrekt eingelesen, da diese tatsächlich in MacRoman codiert ist.

```
Dim f As FolderItem
Dim t as TextInputStream
f = GetFolderItem("Beispiel.txt")
If f<> Nil Then
       t = f.OpenAsTextFile
       t.Encoding = Encodings.MacRoman
       Editfield1.text = t.ReadAll
       t.close
End if
```

In der Zeile, in der die Codierung gesetzt wird, wird das Encodings-Objekt verwendet. Dieses Objekt enthält Funktionen für alle möglichen Codierungen. Die Standardcodierung ist UTF-8, ein spezielles Format von Unicode. Wenn Sie „Encodings“ im Code-Editor eingeben und dann die Tabulatortaste drücken, wird die automatische Vervollständigung des Editors Ihnen alle verfügbaren Codierungen anzeigen.

Sie können die Codierung auch als Parameter für die Methoden Read und ReadAll angeben, statt sie der Encoding-Eigenschaft zuzuweisen.

Schreiben von Textdateien

Wenn Ihre Applikation eine Datei in einer bestimmten Codierung speichern soll, müssen Sie dies beim Speichern angeben. Dieser Fall kann eintreten, wenn die Datei unter einem anderen Betriebssystem oder in einem anderen Land verarbeitet werden soll.

Sie verwenden die ConvertEncoding-Funktion, um Text zwischen verschiedenen Codierungen zu konvertieren. Hier ist ein einfaches Beispiel, das den Text eines EditFields als MacRoman-codiert in einer Datei speichert. Mit Hilfe der ConvertEncoding-Funktion wird der Text vor dem Schreiben in die andere Codierung konvertiert.

```
Dim f As FolderItem
Dim t as TextOutputStream
f = GetFolderItem("Beispiel.txt")
If f<> Nil Then
       t = f.CreateTextFile
       t.Write ConvertEncoding(Editfield1.text, Encodings.MacRoman)
       t.close
End if
```

Wenn eine Applikation eigene Dateien schreibt und liest, brauchen Sie sich darüber keine Gedanken zu machen. Es wird grundsätzlich die Codierung UTF-8 angenommen. Wenn Sie also Dateien schreiben und nichts weiter angeben, werden diese in Unicode geschrieben. Dies gilt auch für das Lesen.

Ermitteln eines einzelnen Zeichens

Wie schon erwähnt, benötigen Sie die Chr-Funktion, wenn Sie ein einzelnes ASCII-Zeichen aus seinem numerischen Wert erzeugen möchten. Die Chr-Funktion funktioniert nur für ASCII-Zeichen im Wertebereich 0 bis 127.

Für andere Werte und Codierungen verwenden Sie die Chr-Methode der TextEncoding Klasse. Um die Chr-Methode verwenden zu können, müssen Sie die Codierung festlegen und einen numerischen Wert angeben, dessen Zeichen Sie ermitteln wollen. Dies sieht dann folgendermaßen aus:

```
Dim s as String
s=Encodings.MacRoman.Chr(170)
```

Wie Sie sehen, wird hier die Chr-Methode für die MacRoman-Codierung aufgerufen. Dadurch wird die Codierung, für die Sie ein Zeichen ermitteln wollen, implizit angegeben. Das Ergebnis wäre ein „™“.

Formatierung von Zahlen, Datums- und Zeitangaben

Zahlen

Zahlen werden formatlos gespeichert. Um Zahlen zu formatieren, übergibt man der Formatierungsfunktion den Zahlenwert und eine Formatierungsanweisung und erhält eine formatierte Zeichenkette zurück:

result=Format(Number, FormatSpec)

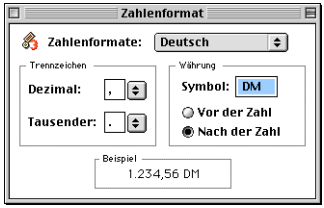
FormatSpec ist eine Zeichenkette, die ein oder mehrere Zeichen enthält, die die Formatierung der Zahl bestimmen. Die Angabe „$###,##0.00“ in FormatSpec führt zur typischen Ausgabe eines Geldbetrages in US-Dollar.

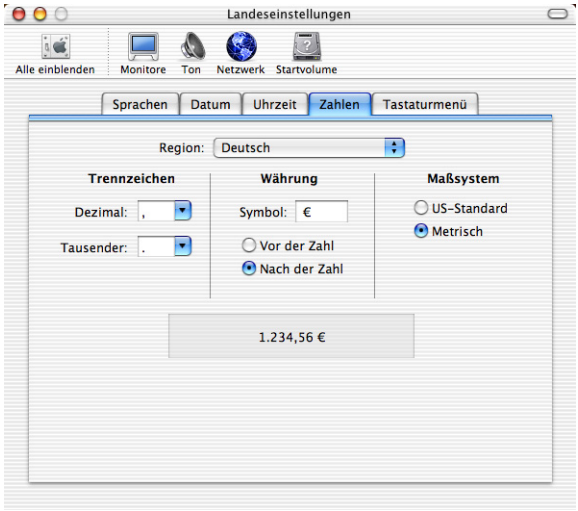
Tabelle 19. In FormatSpec verwendete Zeichen

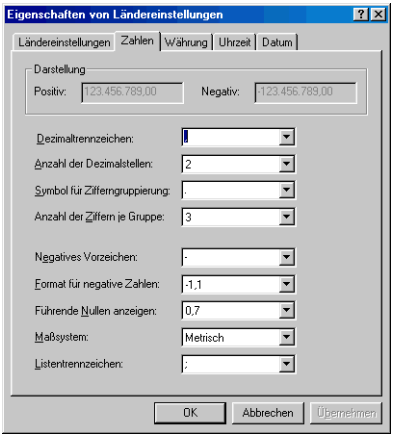
Zeichen	Funktion
#	Platzhalter, an dessen Stelle eine Ziffer des anzuzeigenden Wertes ausgegeben wird, sofern die Zahl lang genug ist.
0	Platzhalter, an dessen Stelle eine Ziffer des anzuzeigenden Wertes ausgegeben wird, sofern die Zahl lang genug ist. Ist an dieser Stelle der Zahl keine Ziffer auszugeben, wird eine 0 ausgegeben.
.	Platzhalter für die Position des Dezimalpunktes. (US-Format)
,	Platzhalter, an dessen Stelle das Komma als Dezimaltrenner erscheinen soll. (US-Format)
%	Ausgabe der Zahl nach Multiplikation mit 100.
(	Ausgabe einer öffnenden Klammer.
)	Ausgabe einer schließenden Klammer.
+	Links von der Zahl wird ein Pluszeichen ausgegeben, wenn es sich um eine positive Zahl handelt, ein Minuszeichen dagegen bei einer negativen Zahl.
–	Links von der Zahl wird ein Minuszeichen ausgegeben, wenn es sich um eine negative Zahl handelt. Bei einer positiven Zahl wird nichts ausgegeben.
E oder e	Ausgabe der Zahl in Exponentendarstellung.
\character	Ausgabe des Zeichens, das auf den Backslash folgt (anstelle des Worts „character“).

Im Classic Mac OS wird das Zeichen, das als Tausender- und Dezimaltrennzeichen verwendet wird, vom Anwender im Zahlenformat-Kontrollfeld festgelegt. Unter Mac OS X werden diese Zeichen bei den Landeseinstellungen im Karteireiter „Zahlen“ eingestellt. Unter Windows stellt man das Zahlenformat bei den Ländereinstellungen ein.

Abb. 207: Kontrollfeld „Zahlenformat" von Mac OS 9, Landeseinstellungen von Mac OS X und Ländereinstellungen von Windows







FormatSpec wendet das angegebene Format auf alle Zahlen an. Wenn Sie unterschiedliche Formate für positive und negative Zahlen und die Null haben möchten, dann geben Sie diese Formatanweisungen einfach durch Semikolon getrennt in FormatSpec an (wie bei den letzten drei Beispielen in der Tabelle:

Tabelle 20. Verschiedene FormatSpecs

Format Syntax	Ergebnis
Format(1.784, "#.##")	1.78
Format(1.3, "#.0000")	1.3000
Format(5, "0000")	0005
Format(.25, "#%")	25%
Format(145678.5, "#.##")	145,678.5
Format(145678.5, "#.##e+")	146e+5
Format(-3.7, "-#.##")	-3.7
Format(3.7, "+#.##")	+3.7
Format(3.7, "#.##; (#.##); \n\u\l\l")	3.7
Format(-3.7, "#.##; (#.##); \n\u\l\l")	(3.7)
Format(0, "#.##; (#.##); \n\u\l\l")	null

Datumsangaben

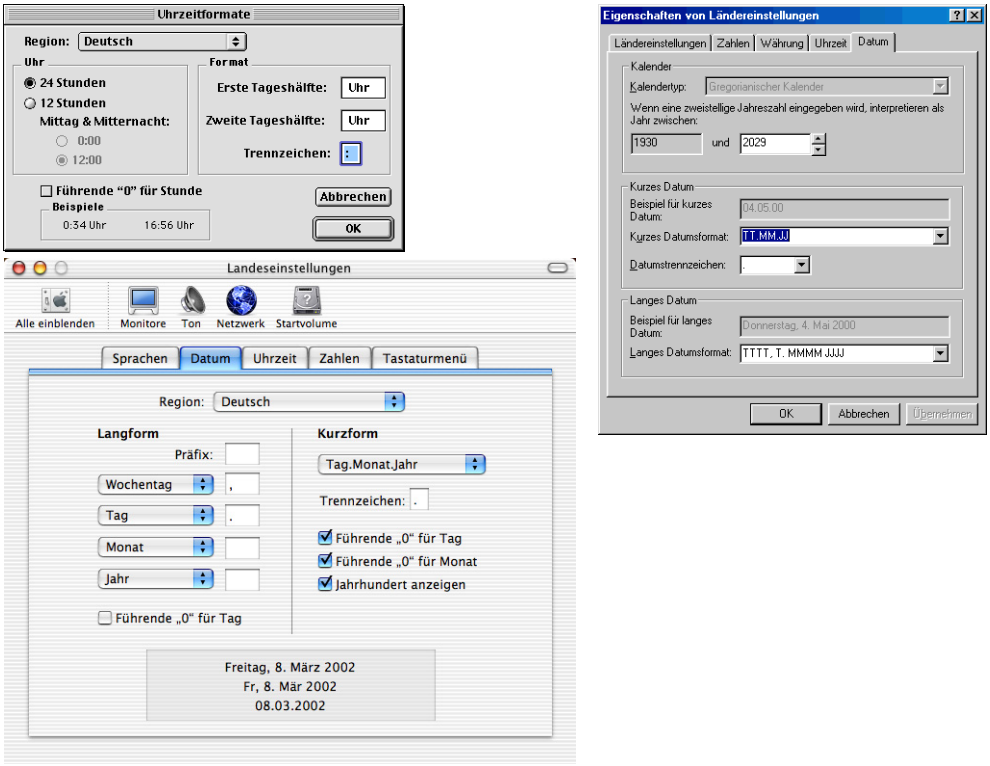
Datumsangaben sind Objekte, deren Eigenschaften das Datum in verschiedenen Varianten enthalten. Die folgende Tabelle zeigt, wie man das Datum in verschiedenen Formaten auslesen kann:

Tabelle 21. Format-Eigenschaften des Date-Objekts

Eigenschaft	Beispiel
ShortDate	31.12.99
LongDate	Freitag, 31. Dezember 1999
AbbreviatedDate	Fre, 31. Dez, 1999

Verschiedene Aspekte des langen und kurzen Datumsformats werden über die Systemeinstellungen kontrolliert. Das Datumsformat wird auf im Classic Mac OS über das Kontrollfeld „Datum und Uhrzeit“, unter Mac OS X im Datum oder Uhrzeit-Karteireiter bei den Landeseinstellungen und unter Windows bei den Ländereinstellungen bei Datum und bei Uhrzeit eingestellt. Man kann die Reihenfolge von Tag, Monat, Jahr als auch die Trennzeichen konfigurieren.

Abb. 208: Die Zeit- und Datumsformateinstellungen der verschiedenen Betriebssysteme



Um das Datum in der jeweiligen Formatierung zu erhalten, legt man einfach ein Datumsobjekt an und greift auf die jeweilige Eigenschaft zu, die das gewünschte Format anbietet. In folgendem Beispiel wird das aktuelle Datum, als LongDate formatiert, einer Variable zugewiesen:

```
Dim today as New Date
Dim theDate as String
theDate=today.LongDate
```

Die TotalSeconds-Eigenschaft eines Date-Objektes ist die Master-Eigenschaft, die Datum und Uhrzeit des Objektes speichert. Die TotalSeconds-Eigenschaft ist definiert als Anzahl der Sekunden seit dem 1.1.1904.

Andere Eigenschaftswerte werden von TotalSeconds abgeleitet. Wenn Sie den Wert der Eigenschaft TotalSeconds ändern, ändern sich die Werte der Eigenschaften von Year, Month, Day, Hour, Minute und Second auf die Sekunde, auf die TotalSeconds zutrifft. Umgekehrt, wenn Sie eine dieser Eigenschaften ändern, ändert sich TotalSeconds ebenfalls entsprechend.

In Windows-Applikationen müssen Sie darauf achten, dass Sie die Day-Eigenschaft nicht auf einen Wert setzen, der größer ist als die Tage des aktuellen Monats. Wenn Sie einen solchen Wert zuweisen möchten, sollten Sie zuerst die Day-Eigenschaft auf einen gültigen Wert für einen beliebigen Monat setzen, danach die restlichen Eigenschaften einstellen und als letztes die Day-Eigenschaft einstellen. Eine andere Möglichkeit unter Windows ist, direkt mit der TotalSeconds-Eigenschaft zu arbeiten und dann die zugehörigen Werte von Year, Month und Day zu ermitteln.

Das Format ist von den Einstellungen im Mac OS abhängig. Es kann also auch eine Ausgabe im amerikanischen Format erfolgen, je nachdem, wie das System konfiguriert ist. Sie sollten sich dessen bewusst sein, dass die Ausgabe, die Ihr Programm vornimmt, auf einem anderssprachigen System automatisch anders aussieht.

Zeitangaben

Zeitangaben werden als Teil eines Datums gespeichert. Datumsobjekte haben zwei Eigenschaften, die die Zeit in unterschiedlichen Formaten liefern.

Tabelle 22. Format-Eigenschaften des Time-Objekts

Eigenschaft	Beispiel
ShortTime	17:40 Uhr
LongTime	17:40:58 Uhr

Um die aktuelle Uhrzeit in einem dieser Formate zu erhalten, erzeugen Sie zunächst ein Date-Objekt und greifen dann auf die jeweilige Eigenschaft zurück.

```
Dim today as Date
Dim Now as String
today=new Date
Now=today.LongTime
```

Genau wie bei den Datums-Formaten kann man verschiedene Aspekte des kurzen und des langen Zeit-Formats bei den entsprechenden Systemeinstellungen konfigurieren.

Suchen mit regulären Ausdrücken

In REALbasic können Sie Text unter Verwendung von regulären Ausdrücken suchen und ersetzen. Reguläre Ausdrücke verwenden eine Meta-Sprache in welcher Sie nach speziellen Zeichen, spezifischen Zeichen (z.B. nur Vokale) und ab bestimmten Positionen (z.B. am Anfang oder Ende einer Zeile) suchen können. Sie definieren in dieser Sprache den String, nach dem gesucht werden soll und (optional) den String, mit der ersetzt werden soll.

Ein regulärer Ausdruck zum Suchen/Ersetzen wird mit den Eigenschaften der RegEx-Klasse festgelegt. Folgende Tabelle listet diese Eigenschaften auf:

Tabelle 23. Eigenschaften der RegEx-Klasse.

Name	Beschreibung
Options	Diese Optionen sind verschiedene Zustände, die Sie für die Engine für reguläre Ausdrücke setzen können. Siehe Tabelle 25.
ReplacementPattern	Dies ist der Ersetze-String, der über die bei regulären Ausdrücken übliche Standard-Notation '\1' oder '$1' Verweise auf bereits gefundene Substrings beinhalten kann. Dieses Muster wird entweder in der Replace-Eigenschaft verwendet oder an die RexExMatch-Klasse übergeben, wenn Search erfolgt ist und wird später mit Replace verwendet, wenn keine Parameter festgelegt werden.
SearchPattern	Dies ist das Muster, nach dem Sie momentan suchen.
SearchStartPosition	Zeichen-Position, an der Sie die Suche starten wollen, wenn der optionale Parameter TargetString zu *Replace* nicht festgelegt wurde. Beachten Sie, dass, wenn Sie ihn setzen, er nur verwendet wird, wenn Sie keinen TargetString festlegen, da das Setzen eines neuen TargetString den Wert zurücksetzt.

Die Methoden der RegEx-Klasse, die in folgender Tabelle aufgelistet sind, werden zum Suchen und Ersetzen verwendet:

Tabelle 24. Methoden der RegEx-Klasse.

Name	Parameter	Beschreibung
Replace	Optional: TargetString als String; SearchStartPosition als Integer	Findet *SearchPattern* in *Target*. Ersetzt den Inhalt von *SearchPattern* mit *ReplacementPattern*. Liefert den resultierenden String. Replace kann den links angegebenen optionalen Parameter erhalten. Liefert einen String zurück.
Search	TargetString als String; SearchStartPosition als Integer	Findet *SearchPattern* in *TargetString* und liefert, falls erfolgreich, ein RexExMatch. Das RexExMatch merkt sich ReplacementPattern, das zum Zeitpunkt der Suche festgelegt wurde.

Mit der RegExOptions-Klasse können Sie folgende Optionen festlegen:

Tabelle 25. Optionen bei den regulären Ausdrücken.

Name	Beschreibung
CaseSensitive	Legt fest, ob Groß-/Kleinschreibung beim Suchen eines Strings berücksichtigt werden soll. Voreinstellung ist False.
DotMatchAll	Normalerweise findet der Punkt alles außer einem Zeilenvorschub. Diese Option erlaubt ihm, Zeilenvorschübe zu finden.
Greedy	Greedy bedeutet, dass die Suche alles vom ersten bis zum Ende des letzten Trennzeichen und alles dazwischen sucht. Wenn Sie z.B. den folgenden Text mit Bold-Tags in HTML finden wollen: The quick brown fox jumped Wenn Sie dieses Muster verwenden: .+ erhalten Sie bei der Suche "quick brown fox", was nicht das ist, was Sie wollten. Daher können Sie *Greedy* ausschalten oder diese Syntax verwenden: .+? und Sie werden "quick" finden, was genau das ist, was Sie wollten.)

Tabelle 25. Optionen bei den regulären Ausdrücken.

Name	Beschreibung
LineEndType	Dies wirkt sich auf die aktuelle Session der regulären Ausdrücke aus) Es hat keinen Effekt auf SearchPatterns, wenn *TreatTargetAsOneLine* True ist. Ändert die Art, wie \n für ReplacementPatterns expandiert wird 0 = beliebiges Zeilenende (Mac oder Win32 oder Unix) 1 = Default für die Plattform (wenn auf einem Mac, dann 2) (wenn unter Win32, dann 3) 2 = Mac ASCII 13 oder \r 3 = Win32 ASCII 10 oder \n 4 = Unix ASCII 10 oder \n
MatchEmpty	Legt fest, ob es Mustern erlaubt ist, leere Strings zu finden.
ReplaceAllMatches	Legt fest, ob jedes Auftreten des Musters ersetzt werden soll.
StringBeginIsLineBegin	Legt fest, ob der Anfang eines Strings als der Anfang einer Zeile zählt.
TreatTargetAsOneLine	Ignoriert interne Zeilenumbrüche, um '^' und '$' zu finden

Wenn Sie beim Suchen unter Verwendung von regulären Ausdrücken einen Treffer landen, können Sie die Eigenschaften der RegExMatch-Klasse verwenden, um den passenden String (oder die passenden Strings) zu erhalten und optional das Ergebnis mit einem String Ihrer Wahl ersetzen. Die Eigenschaften der RegExMatch-Klasse sind folgende:

Tabelle 26. Eigenschaften der RexExMatch-Klasse.

Name	Beschreibung
SubExpressionCount	Anzahl der verfügbaren SubExpressions der gerade erfolgten Suche. Die regulären Ausdrücke von REALbasic unterstützten sowohl \number als auch $number für SubExpressions. SubExpressions erlauben das Ersetzen von Teilen des Musters (Pattern).
SubExpressionString	Liefert SubExpression als einen String für die angegebene matchNumber. 0 liefert den gesamten MatchString (das implizierte 0te subExpression) und 1 ist die erste echte subExpression.
SubExpressionStart	Liefert die Startposition von der durch matchNumber festgelegten subExpression.
Replace	Ersetzt das gefundene Ergebnis in der Weise, die durch *ReplacementPattern* festgelegt wird. Wenn kein *ReplacementPattern* festgelegt ist, verwendet es das ReplacementPattern, das im RegEx-Objekt zur Zeit der Suche festgelegt wurde.

Weitere Informationen finden Sie in der Sprachreferenz.

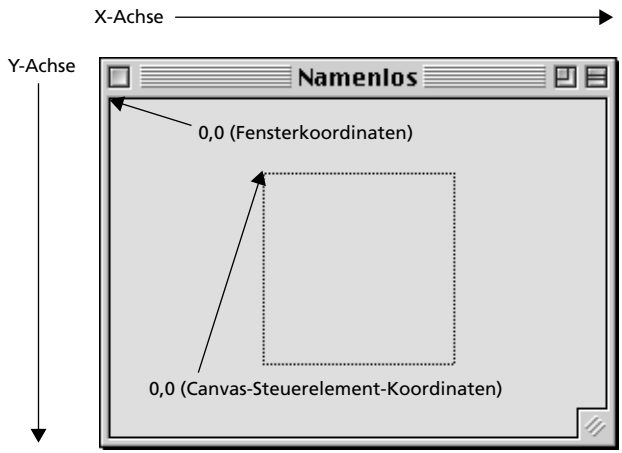
Bilder und Grafik

Das Koordinatensystem

Bei den meisten grafischen Methoden muss man angeben, an welcher Stelle innerhalb des Canvas-Steuerelements gezeichnet werden soll. Diese Angabe macht man bezogen auf das Koordinatensystem, das man sich als unsichtbares Gitter aus horizontalen und vertikalen Linien mit einem Pixel Abstand vorstellen kann. Wenn Sie Koordinatensysteme nur aus der Mathematik kennen, dann erwarten Sie den Ursprung (0,0) vielleicht in der Mitte des Fensters. Das wäre allerdings sehr unpraktisch, daher liegt der Ursprung immer in der linken oberen Ecke der Zeichenfläche. Der Ursprung des gesamten Bildschirms ist die linke obere Ecke des Bildschirms. Bei einem Fenster ist dies dessen linke obere Ecke und für ein Steuerelement ist es wiederum dessen linke obere Ecke. Die Werte auf der X-Achse (Horizontalachse) erhöhen sich von links nach rechts und auf der Y-Achse (Vertikalachse) werden sie von oben nach unten größer.

Ein Punkt innerhalb eines Fensters mit den Koordinaten 10,20 befindet sich also 10 Punkte rechts vom linken Rand und 20 Punkte unter dem oberen Rand. Wenn Sie innerhalb eines Canvas-Steuerelements arbeiten, befindet sich der Punkt 10,20 von der linken Kante des Steuerelements 10 Pixel und von der oberen Kante 20 Pixel entfernt.

Abb. 209: Das X,Y-Koordinatensystem



Darstellen von Bildern in einem Fenster

Benutzen des gesamten Fensters zur Darstellung

Wenn Sie ein Bild in einem Fenster darstellen wollen, dann können Sie dazu die Backdrop-Eigenschaft des Fensters verwenden. Hier kann ein Bild abgelegt werden, das dann im Fenster angezeigt wird und das auch automatisch hinter eventuell im Fenster platzierten Steuerelementen liegt. Die Eigenschaft Backdrop hat normalerweise den Wert „kein" (None). Es gibt mehrere Möglichkeiten, ein Bild der BackDrop-Eigenschaft eines Fensters zuzuweisen. Auf dem Macintosh ist der einfachste Weg, die Bild-Datei aus dem Finder in das Fenster in der Entwicklungsumgebung zu ziehen. Wenn Sie das machen, passieren mehrere Dinge: Das Bild erscheint im Fenster in der Entwicklungsumgebung, der Name der Bilddatei wird dem Projekt hinzugefügt und es wird der BackDrop-Eigenschaft im Eigenschaftenfenster des Fensters zugewiesen.

Sie können ein Bild auch der BackDrop-Eigenschaft eines Fensters zuweisen, indem Sie die Bild-Datei in Ihr Projektfenster ziehen und denn den Namen des Bildes als BackDrop-Eigenschaft im Eigenschaftenfenster auswählen. In einem Open-Event-Handler eines Fensters können Sie z.B. schreiben: Backdrop=*Bildname,* wobei *Bildname* der Name des Bildes ist, wie er im Projektfenster erscheint.

Zusätzlich können Sie zur Laufzeit die BackDrop-Eigenschaft setzen, indem Sie ein Bild laden oder mit den Methoden der Graphics-Klasse neu zeichnen. Folgendes Beispiel zeigt, wie der Benutzer mit einer ganz normalen Dateiauswahlbox PICT, JPEG oder GIF-Dateien wählen kann, die dann als Backdrop-Bild im Fenster angezeigt werden:

```
Dim f as FolderItem
f=GetOpenFolderItem("image/gif;image/jpeg;image/x-pict")
If f<> Nil Then
 Backdrop=f.OpenAsPicture
End If
```

Achtung: Dieses Beispielprogramm funktioniert nur dann, wenn Sie die Dateitypen image/gif, image/jpeg und image/x-pict in der Dialogbox **Bearbeiten/Dateitypen** definiert haben.

Nachdem Sie der BackDrop-Eigenschaft ein Bild zugewiesen haben, können Sie die Größe des Fensters automatisch an die Größe des Bildes anpassen, indem Sie den Eigenschaften Height (Höhe) und Width (Breite) des Fensters jeweils die Eigenschaften Height (Höhe) und Width (Breite) des Bildes zuweisen:

```
width=Backdrop.width
height=Backdrop.height
```

Das Neuzeichnen des Backdrop-Bildes übernimmt REALbasic automatisch.

Darstellung in einem Teilbereich des Fensters

Wenn Sie das Bild nur in einem Teilbereich des Fensters anzeigen wollen, können Sie ein ImageWell-Steuerelement verwenden. Ein ImageWell ist einem Canvas-Steuerelement sehr ähnlich, außer dass es keine Zeichenwerkzeuge besitzt: Sie können nur ein Bild darstellen, das schon an anderer Stelle erzeugt wurde.

Um ein Bild der Image-Eigenschaft eines ImageWells zuzuweisen, draggen Sie es einfach aus dem Finder in das Projektfenster und weisen es dann der Image-Eigenschaft des ImageWell unter Verwendung seines Eigenschaftenfensters zu.

Sie können zur Laufzeit ein Bild hinzufügen, indem Sie es mittels Code laden. Folgendes Code-Beispiel zeigt eine Dateiauswahlbox an, die die Auswahl einer PICT-, JPEG-, oder GIF-Datei erlaubt und zeigt diese dann im ImageWell an. Voraussetzung ist, dass die als Parameter für die Dateiauswahlbox verwendeten Dateitypen in Ihrem Projekt unter **Bearbeiten/Dateitypen** definiert wurden.

```
dim f as FolderItem
f=GetOpenFolderItem("image/x-pict;image/jpeg;image/gif")
if f <> Nil then
 ImageWell1.Image=f.OpenAsPicture
end if
```

Sie können dem Anwender auch erlauben, ein Bild-Dokument aus dem Finder auf ein ImageWell zu draggen, ohne dass die Dateiauswahlbox verwendet wird. Im Open-Event-Handler des ImageWells erlauben Sie das Droppen einer Datei mit folgenden Zeilen:

```
me.acceptfileDrop("image/x-pict")
me.acceptfileDrop("image/jpeg")
me.acceptfileDrop("image/gif")
```

Im DropObject-Event-Handler des ImageWells wird folgender Code verwendet:

```
Sub (DropObject (Obj as DragItem)
 If Obj.FolderItemAvailable then
  Me.Image=Obj.FolderItem.OpenAsPicture
 End if
End Sub
```

Hinweis: Dieser Code unterstützt nur PICT-Dateien (Mac) und .BMP-Dateien (Windows), so lange kein QuickTime auf dem Rechner installiert wurde.

Die andere Möglichkeit ist, ein Canvas-Steuerelement zu verwenden, um ein Bild in einem Teilbereich des Fensters anzuzeigen. Dieses Steuerelement stellt eine Zeichenfläche zur Verfügung, in der Sie Zeichenoperationen ausführen können, die aber auch Events entgegennimmt. Daher können Sie ein Canvas-Steuerelement auch verwenden, um dem Benutzer eine Interaktion zu gestatten. Mit der Backdrop-Eigenschaft des Canvas-Steuerelements können Sie ein vorhandenes Bild anzeigen. Wie beim Fenster schon gezeigt, kann das manuell in der Entwicklungsumgebung geschehen oder mittels Laden eines Bildes und Zuweisen an die Backdrop-Eigenschaft. Hier ein Beispiel analog zu dem obigen:

```
Dim f as FolderItem
f=GetOpenFolderItem("image/x-pict")
If f<> Nil Then
 Me.Backdrop=f.OpenAsPicture
End If
```

Sie können Drag&Drop auf ein Canvas-Steuerelement in der gleichen Weise wie für ein ImageWell realisieren. Der Vorteil bei der Verwendung eines Canvas-Steuerelements ist, dass Sie das Bild mit den Methoden der Graphics-Klasse unter Ver-

wendung des Paint-Event-Handlers des Canvas-Steuerelements verändern können. Dies wird im folgenden Abschnitt beschrieben.

Zeichnen von Bildern

Sie können in Ihrem Programm Bilder mit den Methoden der Graphics-Klasse zeichnen. Ein Graphics-Objekt ist ein Objekt im Speicher, das ein Bild enthält. Die Objekte Window und Canvas haben ein Paint-Event. Dieses wird immer dann ausgeführt, wenn das Window- oder Canvas-Objekt neu gezeichnet werden muss. Wird beispielsweise ein Fenster geöffnet, dann wird das Paint-Event aktiv, denn der Inhalt des Fenster muss gezeichnet werden. Jedes Canvas-Element im Fenster wird ebenfalls ein Paint-Event ausführen, da das Canvas-Objekt Bestandteil des zu zeichnenden Fensters sind. Diese Events werden auch dann ausgeführt, wenn ein Teil des Fensters verdeckt war und nun gezeichnet werden muss, weil ein anderes Fenster verschoben wurde.

Das Paint-Event erhält ein Graphics-Objekt als Parameter. Bei Beendigung des Paint-Events wird dieses Objekt im Window- oder Canvas-Objekt gezeichnet. Im Window- oder Canvas-Objekt zeichnen Sie mittels der Zeichen-Methoden des jeweiligen Graphic-Objekts.

Anzeigen von Bildern

Bilder werden in Graphics-Objekten mittels der DrawPicture-Methode der Graphics-Klasse gezeichnet. Dieser Methode wird das zu zeichnende Bild übergeben und die Koordinaten, an denen es innerhalb des Graphics-Objekts ausgegeben werden soll. Im folgenden Beispiel werden mit dem Paint-Event zwei Bilder (BartPict und LisaPict) nebeneinander gezeichnet. Die beiden Bilder müssen dazu zuvor in Ihr Projektfenster gezogen werden:

```
Sub Paint(g As Graphics)
 g.DrawPicture BartPict, 0,0
 g.DrawPicture LisaPict,BartPict.Width, 0
End Sub
```

Kopieren von Bildteilen

Ausschnitte von Bildern lassen sich mit der DrawPicture Methode aus der Graphics-Klasse in ein Graphics-Objekt kopieren. Dies erreicht man mit optionalen Parametern der DrawPicture-Methode, mit denen ein Teilbereich eines Bildes, der gezeichnet werden soll, definiert werden kann. Man kann die Ausgangskoordinaten sowie die Höhe und Breite des Teilbereiches angeben.

Im folgenden Beispiel wird ein Bereich von 20 mal 20 Pixeln des Quellbildes gezeichnet, der zehn Pixel vom linken Rand und zehn Pixel vom oberen Rand entfernt beginnt. Der Ausschnitt wird dann fünf Pixel vom linken Rand und fünf Pixel vom oberen Rand entfernt in das Canvas- oder Window-Objekt gezeichnet:

```
Sub Paint(g As Graphics)
 g.DrawPicture Lisa,5,5,Lisa.width,Lisa.height,10,10,20,20
End Sub
```

Skalieren von Bildern

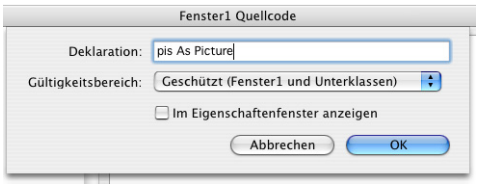
Die Methode DrawPicture der Graphics-Klasse erlaubt auch die Skalierung von Bildern. Dazu müssen alle DrawPicture-Parameter angegeben werden. Die Skalierung erfolgt einfach dadurch, dass für das auszugebende Bild eine Breite oder Höhe angegeben wird, die von der Originalbreite oder -höhe abweicht. Im Beispiel wird das Bild doppelt so groß ausgegeben, wie es im Original ist:

```
Sub Paint(g As Graphics)
 Dim w,l as integer
 w=lisa.width
 l=lisa.height
 g.DrawPicture lisa, 0,0,w*2,l*2,0,0,lisa.width,lisa.height
End Sub
```

Es folgt ein Beispiel, das ein importiertes Bild automatisch auf die Größe eines Canvas-Steuerelements skaliert.

Angenommen, Sie haben ein Fenster mit einem Canvas-Steuerelement und einem PushButton (oder einem anderen Steuerelement), der den Code für das Importieren und das Skalieren auslöst.

Deklarieren Sie eine Variable für das Elternfenster, die das skalierte Bild enthält:



Im ActionEvent des Push-Buttons verwenden Sie den folgenden Code, um das Bild zu importieren und zu skalieren:

```
Dim f As FolderItem
Dim p As Picture
Dim maxWidth, maxHeight As Integer
Dim factor As Double

maxWidth = Canvas1.width
maxHeight = Canvas1.height

f = GetOpenFolderItem("PICT")
If f <> nil then
 p = f.OpenAsPicture
end if

factor = Min( maxWidth / p.Width, maxHeight / p.Height )
factor = Min( factor, 1.0 ) // (nicht skalieren, wenn es zu klein ist!)

pic = NewPicture( p.Width * factor, p.Height * factor, 32 )
pic.graphics.DrawPicture p, 0,0,pic.width,pic.height, 0,0,p.width,p.height

Canvas1.Refresh
```

(Dieses Codebeispiel geht davon aus, dass Sie eine PICT-Datei importieren möchten und dass Sie den Dateityp PICT in der Dateitypen-Dialogbox definiert haben).

Die Werte von **maxWidth** und **maxHeight** entsprechen in diesem Beispiel der Größe des Canvas-Steuerelementes. Sie können natürlich auch andere Werte nehmen. Die DrawPicture-Methode der Graphics-Klasse verwendet die folgenden Parameter:

Table 27:

Parameter	Typ	Beschreibung
Image	Picture	Das zu zeichnende Bild an der durch X und Y angegebenen Position.
X	Integer	X-Koordinate innerhalb des Steuerelements der linken Bildseite. Voreinstellung ist null.
Y	Integer	Y-Koordinate innerhalb des Steuerelementes der oberen Bildbegrenzung.Voreinstellung ist null.
DestWidth	Integer	Breite des Bildes im Steuerelement.
DestHeight	Integer	Höhe des Bildes im Steuerelement.
SourceX	Integer	X-Koordinate des Bildes, das Sie kopieren. Voreinstellung ist null
SourceY	Integer	Y-Koordinate des Bildes, das Sie kopieren. Voreinstellung ist null.

Table 27:

Parameter	Typ	Beschreibung
SourceWidth	Integer	Breite des Bildes, das Sie kopieren möchten. Voreinstellung ist die Breite des Bildes.
SourceHeight	Integer	Höhe des Bildes, das Sie kopieren möchten. Voreinstellung ist die Höhe des Bildes.

Zum Abschluss müssen Sie zum PaintEvent des Canvas-Steuerelements den Code hinzufügen, der der Backdrop-Eigenschaft des Canvas-Steuerelements das skalierte Bild „pic“ zuweist.

```
If pic <>Nil then
 g.DrawPicture pic,0,0
End if
```

Standard-Piktogramme in Dialogen

Die MsgBox-Funktion von REALbasic zeichnet normalerweise einfach ein Sprechblasen-Icon und einen OK-Knopf in eine Messagebox. Das Sprechblasen-Icon ist aber nicht ganz passend, wenn der Benutzer gewarnt werden soll. Ist der Benutzer im Begriff etwas zu tun, das mit dem Verlust von Daten enden könnte (wie beispielsweise das Verlassen des Programms ohne vorheriges Sichern), sollte das Warn-Icon angezeigt werden. Das Stop-Icon verwendet man, wenn der Benutzer etwas machen will, das gerade nicht geht, wie etwa das Speichern des Dokuments auf einem schreibgeschützten Medium.

Abb. 210: Sprechblasen-, Warn- und Stop-Icons unter Windows XP und Mac OS X

Hinweis: Unter Mac OS Classic kann das Aussehen dieser drei Icons je nach Einstellung im Erscheinungsbild-Kontrollfeld variieren. Unter Windows können diese Icons je nach Desktopdarstellung unterschiedlich aussehen.

Die Graphics-Klasse bietet dazu die Methoden DrawNoteIcon, DrawCautionIcon und DrawStopIcon, mit denen diese Icons einfach in einem Canvas- oder Windows-Objekt angezeigt werden können. Man sollte diese Methoden verwenden, da die Icons von einer Version des Betriebssystems zur nächsten wechseln können und auch auf einer anderen Betriebssystemplattform anders aussehen. So wird immer das Icon gezeigt, das zum System passt.

So wird im Paint-Event eines Canvas-Objekts ein Sprechblasen-Icon gezeichnet:

```
Sub Paint(g As Graphics)
 g.DrawNoteIcon 0,0
End Sub
```

Pixel zeichnen

Mit der Pixel-Eigenschaft können Sie die Farbe eines beliebigen Punktes in einem Graphics-Objekt setzen oder auslesen. Dazu geben Sie die Koordinaten des Pixels an und setzen oder lesen die Farbe.

Das Beispiel zeichnet Pixel an zufällig gewählten Koordinaten mit zufälligen Farben in einem Graphics-Objekt, bis der Benutzer die ESC-Taste oder ⌘-„.“ (Macintosh) drückt:

```
Sub Paint(g As Graphics)
 Dim c as Color
 Do
 c=Rgb(Rnd*255,Rnd*255,Rnd*255)
 g.Pixel(Rnd*me.Width,Rnd*me.Height)=c
 Loop until UserCancelled
End Sub
```

Nun ein Beispiel, wie man die Farbe des Pixels unter der Mausspitze über einem Canvas-Objekt ausliest und diese auf ein anderes Canvas-Objekt (namens PixelColor) überträgt:

```
Sub MouseMove(X As Integer, Y As Integer)
 Dim c as Color
 c=Me.Graphics.Pixel(X,Y)
 PixelColor.Graphics.ForeColor=c
 PixelColor.Graphics.FillRect 0,0,PixelColor.Width,PixelColor.Height
End Sub
```

Zeichnen von Linien

Linien werden mit der DrawLine-Methode aus der Graphics-Klasse gezeichnet. Die Farbe der Linie ist in der Eigenschaft ForeColor des Graphics-Objekts gespeichert, in dem gezeichnet wird. Zum Zeichnen der Linie muss man der Methode die Anfangs- und die Endkoordinaten der Linie übergeben.

Dieses Beispiel zeichnet ein Gitter innerhalb eines Canvas-Objekts oder in einem Fenster. Die Größe jedes Kästchens wird durch die Variable boxSize vorgegeben:

```
Sub Paint(g as Graphics)
 Dim i, boxSize as Integer
 boxSize=10
 For i=boxSize to Me.Width Step boxSize
 g.DrawLine i,0,i,Me.Height
 Next
 For i=boxSize to Me.Height Step boxSize
 g.DrawLine 0,i,Me.Width,i
 Next
End Sub
```

Die Stärke der Linie wird durch die Eigenschaften PenHeight und PenWidth des Graphics-Objekts vorgegeben.

Zeichnen von Ovalen

Mit DrawOval und FillOval bietet die Graphics-Klasse Methoden zum Zeichnen von Ovalen. Für beide sind die gleichen Parameter erforderlich, nämlich die X/Y-Koordinaten, an denen das Oval gezeichnet werden soll und die Breite und Höhe. Beide Methoden zeichnen ein Oval in der ForeColor-Farbe des Graphics-Objekts in der Linienstärke, die in PenWidth und PenHeight vorgegeben sind. DrawOval zeichnet die Ränder des Ovals, während FillOval die Fläche des Ovals zusätzlich mit der ForeColor füllt.

So wird ein Oval in ein Canvas- oder Window-Objekt gezeichnet:

```
Sub Paint(g as Graphics)
 g.DrawOval 0,0,50,75
End Sub
```

Zeichnen von Rechtecken

Rechtecke werden mit den Methoden DrawRect, FillRect, DrawRoundRect und FillRoundRect der Graphics-Klasse gezeichnet. Diese Methoden benötigen alle die linke obere Ecke des zu zeichnenden Rechtecks als X/Y-Koordinate und seine Breite und Höhe. ForeColor, PenWidth und PenHeight geben Farbe und Linienstärke an.

RoundRectangles sind Rechtecke mit gerundeten Ecken. Daher wird für DrawRoundRect und FillRoundRect noch eine Angabe zur Breite und Höhe der Rundung in den Ecken des Rechtecks benötigt.

DrawRect und DrawRoundRect zeichnen die Begrenzungslinien der Rechtecke, während FillRect und FillRoundRect gefüllte Rechtecke zeichnen.

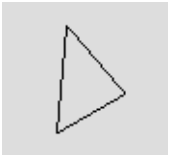
Zeichnen von Polygonen

Polygone werden mit den Funktionen DrawPolygon und FillPolygon der Graphics-Klasse gezeichnet. Dazu wird ein Array übergeben, das jeden Punkt des Polygons enthält. Das Array beginnt mit dem Element 1. Elemente mit ungeraden Nummern enthalten die X-Koordinate, während Elemente mit geraden Nummern die Y-Koordinate enthalten. Das bedeutet, dass Element 1 die X-Koordinate des ersten Polygonpunkts und Element 2 seine Y-Koordinate enthält. Nehmen wir folgendes Array an:

Tabelle 28. Ein Array mit Werten für ein Polygon

Element Nr.	Wert
1	10
2	5
3	40
4	40
5	5
6	60

Wird dieses Array an die DrawPolygon oder FillPolygon-Methode übergeben werden folgende Linien gezeichnet: Von 10,5 nach 40,40, dann von 40,40 nach 5,60 und von 5,60 zurück zu 10,5. Da dieses Polygon nur drei Koordinatenpaare enthält, entsteht ein Dreieck:



Der dazugehörende Code für ein Paint-Event sieht so aus:

```
Sub Paint(g As Graphics)
 Dim points(6) as Integer
 points(1)=10
 points(2)=5
 points(3)=40
 points(4)=40
 points(5)=5
 points(6)=60
 g.DrawPolygon points
End Sub
```

FillPolygon zeichnet dasselbe Polygon, allerdings mit gefülltem Innenraum:

Eigene Steuerelemente mit einem Canvas-Objekt realisieren

Steuerelemente, die wie PushButtons ein grafisches Interface verwenden, mit dem der Anwender interagieren kann, sind nichts anderes als Bilder, die mit einem entsprechenden Unterprogramm gezeichnet werden. Daher kann man mit einem Canvas-Objekt sehr leicht Steuerelemente entwerfen, die nicht in REALbasic eingebaut sind.

Angenommen, Sie möchten ein sehr simples Steuerelement zeichnen, das einfach aus einem Rechteck besteht, dessen Farbe von Schwarz nach Weiß wechselt, wenn man es anklickt. Da das Rechteck seine Farbe jeweils wechseln soll, wenn

der Benutzer es anklickt, gehört der Code dazu in den MouseDown-Event-Handler des Canvas-Objekts. Das Programm muss prüfen, ob das Rechteck weiß ist und es dann schwarz einfärben oder im umgekehrten Fall weiß. Dazu muss man nur die Farbe eines beliebigen Pixels in dem Objekt prüfen, wofür die Rgb-Funktion verwendet wird. Übergibt man dieser eine 0 für alle Parameter, dann ergibt das die Farbe weiß. Schwarz resultiert aus dem Übergeben von 255 für alle Parameter. Später werden Sie noch mehr über Farben erfahren. Für den MouseDown-Handler verwendet man folgendes Programm:

```
Function MouseDown(X As Integer, Y As Integer) As Boolean
 If Me.Graphics.Pixel(X,Y)=Rgb(0,0,0) Then
 Me.Graphics.ForeColor=Rgb(255,255,255)
 Else
 Me.Graphics.ForeColor=Rgb(0,0,0)
 End If
 Me.Graphics.FillRect Me.Left,Me.Top,Me.Width,Me.Height
End Function
```

Es wird überprüft, ob das Pixel, auf den der Benutzer geklickt hat, weiß ist. Ist dies der Fall, dann wird die ForeColor-Eigenschaft des Canvas-Objekts, das allgemeingültig durch die Me-Funktion repräsentiert wird, auf schwarz gesetzt und andernfalls auf weiß. Danach wird die Methode FillRect für das Canvas-Objekt aufgerufen, um es mit der ForeColor zu füllen.

Das Steuerelement ist noch nicht ganz fertig. Wenn es neu gezeichnet werden muss (zum Beispiel beim ersten Öffnen des Fensters) wird das Paint-Event aufgerufen. Da hier kein Code vorhanden ist, wird das Element auch nicht gezeichnet. Daher muss eine leicht modifizierte Variante der Routine, die für den MouseDown-Handler verwendet wurde, eingesetzt werden:

```
Sub Paint(g As Graphics)
 If g.Pixel(0,0)=Rgb(0,0,0) Then
 g.ForeColor=Rgb(255,255,255)
 Else
 g.ForeColor=Rgb(0,0,0)
 End If
 g.FillRect Me.Left,Me.Top,Me.Width,Me.Height
End Sub
```

Da der Paint-Event-Handler mittels des Parameters g eine Referenz auf das Canvas-Objekt bekommt, kann man den Code etwas allgemeingültiger fassen und „g“ statt „me.graphics“ verwenden. Da der Benutzer in diesem Fall nirgends klickt, muss ein Pixel angegeben werden, dessen Farbe geprüft wird. Wir haben im Beispiel das Pixel 0,0 gewählt.

Im „Wiederverwendbare Objekte durch Klassen“ auf Seite 304 werden Sie erfahren, wie man erheblich komplexere Steuerelemente mittels Klassen definieren kann.

Das Arbeiten mit Vektorgrafiken

Die REALbasic-Programmiersprache umfasst eine Reihe von Klassen, die Ihnen das Erzeugen, Öffnen und Sichern von Vektorgrafiken ermöglichen. Eine Vektorgrafik ist (im Gegensatz zur Bitmapgrafik) vollständig aus primitiven Objekten – Linien, Rechtecken, Text, Kreisen und Ovalen usw. – zusammengesetzt, die ihre Identität in der Grafik beibehalten. Sie lösen sich nicht auf und werden nicht Teil einer undifferenzierbaren Bitmap. Die Object2D-Klasse ist die Basisklasse für alle Klassen, mit denen sich primitive Objekte erzeugen lassen. Sie sind in der folgenden Tabelle aufgelistet:

Tabelle 29. Klassen zum Zeichnen von Vektorgrafiken

Klasse	Beschreibung
ArcShape	Zeichnet den Bogen eines Kreises.
CurveShape	Zeichnet gerade Linien oder Kurven unter Verwendung von einem oder mehreren Ankerpunkten.
FigureShape	Zeichnet Polygone, die (optional) gekrümmte Seiten enthalten können.

Tabelle 29. Klassen zum Zeichnen von Vektorgrafiken

Klasse	Beschreibung
OvalShape	Zeichnet Kreise und Ovale.
PixMapShape	Importiert eine Bitmapgrafik in die Vektorzeichnung.
RectShape	Zeichnet Quadrate oder Rechtecke.
RoundRectShape	Zeichnet Quadrate oder Rechtecke mit abgerundeten Kanten (Unterklasse der RectShape-Klasse).
StringShape	Zeichnet String-Text gemäß Font, Fontgröße und Stil.

Da jede der in der obigen Tabelle abgebildeten Klassen sich von der Objekt2D-Klasse ableitet, können Sie die Eigenschaften der Objekt2D-Klasse dazu verwenden, das Objekt zu zeichnen. Sie werden in der folgenden Tabelle aufgelistet:

Tabelle 30. Eigenschaften der Objekt2D-Klasse

Name	Beschreibung
Border	Stärke der Umrandung, reicht von 0 (transparent) bis 100 (undurchsichtig). Voreinstellung ist 0.
BorderColor	Farbe der Umrandung.
BorderWidth	Die Breite der Umrandung in Punkten. Die Voreinstellung beträgt 1.
Fill	Die Farbdichte des eingeschlossenen Teils, reicht von 0 (transparent) bis 100 (undurchsichtig). Voreingestellt ist 100.
FillColor	die Füllfarbe einer Vektorform.
Rotation	Rotation im Uhrzeigersinn, im Bogenmaß, um den Punkt X,Y.
Scale	Skalierungsfaktor relativ zur Originalgröße des Objekts.
X	Horizontale Position des Zentrums oder des Hauptankerpunktes.
Y	Vertikale Position (ausgehend von der top-Eigenschaft nach unten) des Zentrums oder des Hauptankerpunktes.

Jede Klasse besitzt weitere Eigenschaften, die die individuelle Form oder die Eigenschaften des jeweiligen Objekts betreffen. Die RectShape-Klasse beispielsweise besitzt Eigenschaften für die Höhe und Breite des Rechtecks. Die RoundRectShape-Klasse besitzt zusätzliche Eigenschaften, die die Höhe und Breite der Kantenabrundungen sowie die Anzahl der verwendeten geraden Linien für die abgerundeten Ecken genauer festlegen. Weitere Informationen zu den einzelnen Eigenschaften finden Sie in der Sprachreferenz.

Vektorobjekte zeichnen und anzeigen

Sie zeichnen ein Vektorobjekt, indem Sie es ganz einfach instanziieren und seine Eigenschaften festlegen. Der folgende Code zeichnet ein RoundRectShape:

```
Dim r as RoundRectShape
r=New RoundRectShape
r.width=120
r.height=120
r.border=100
r.bordercolor=RGB(0,0,0)//schwarze Umrandung
r.fillcolor=RGB(255,102,102)
r.cornerHeight=15
r.cornerWidth=15
r.borderwidth=2.5
```

Das einzige „Problem“ mit dieser Vektorform ist, dass diese nirgendwo auftaucht. Sie ist lediglich definiert und bereit zum Gebrauch. Sie müssen noch einen Befehl hinzufügen, der das Vektor-Objekt in einem anderen Objekt, also einem Fenster oder ein Steuerelement, das Grafik darstellen kann (z.B. Canvas), anzeigt.

Im Paint-Event eines Steuerelements oder Fensters ist der Code zum Darstellen von Vektor-Objekten gut aufgehoben. Dem Paint-Event-Handler wird ein Graphics-Objekt übergeben und Sie müssen lediglich die DrawObject-Methode der Graphics-Klasse aufrufen, um das Objekt darzustellen. Der Zugriff auf die Graphics-Klasse erfolgt über den Parameter g. Die DrawObject-Methode akzeptiert drei Parameter. Das darzustellende Objekt und die X- und Y-Position, an der es gezeichnet werden soll.

Der folgende Code zeichnet ein fertiges Vektor-Objekt in einem Fenster.

```
Dim r as New RoundRectShape
r.width=120
r.height=120
r.border=100
r.bordercolor=RGB(0,0,0) //black border
r.fillcolor=RGB(255,102,102)
r.cornerHeight=15
r.cornerWidth=15
r.borderwidth=2.5
g.DrawObject r,100,100 //draw at 100,100
```

Vektor-Objekte können aus mehreren einzelnen Vektor-Objekten zusammengesetzt werden. Dieses zusammengesetzte Objekt ist dann ein Group2D-Objekt. Wenn Sie mit der Zusammenstellung fertig sind, können Sie das Objekt über die DrawObject-Methode darstellen.

Der folgende Code zeigt, wie so etwas aussehen kann. Wir haben dem letzten Beispiel ein weiteres RoundedRect-Shape-Objekt hinzugefügt und es um 20 Pixel nach rechts und unten verschoben. Die zwei Append-Anweisungen erzeugen das Group2D-Objekt.

```
Dim r as New RoundRectShape
Dim s as New RoundRectShape
Dim group as New Group2D

r.width=120
r.height=120
r.border=100
r.bordercolor=RGB(0,0,0) // Schwarzer Rahmen
r.fillcolor=RGB(255,102,102)
r.cornerHeight=15
r.cornerWidth=15
r.borderwidth=2.5

s.width=120
s.height=120
s.border=100
s.bordercolor=RGB(0,0,0) // Schwarzer Rahmen
s.fillcolor=RGB(255,102,102)
s.cornerHeight=15
s.cornerWidth=15
s.borderWidth=2.5

s.x=r.x+20  // s um 20 Pixel nach rechts verschieben
s.y=r.y+20  // s um 20 Pixel nach unten verschieben

group.append r
group.append s
g.drawObject group,100,100
```

Wenn Sie einer Bitmapgrafik eine Vektorgrafik hinzufügen wollen, können Sie dies ebenfalls, indem Sie beide über die Append-Methode zu einem Group2D-Objekt zusammenfügen.

In dem folgenden Beispiel werden ein StringShape- und ein PixmapShape-Objekt zu einem Group2D-Objekt zusammengefügt. Die Grafik h1 wurde zuvor in das Projektfenster gezogen.

```
Dim px As PixmapShape
Dim s As StringShape
Dim d As New Group2D

px=New PixmapShape(h1) //h1 ist eine Grafik im Projektfenster
d.append px

s=New StringShape
s.y=70
s.Text="Das ist es, was ich ein ECHTES Auto nenne!"
s.TextFont="Helvetica"
s.Bold=true
d.append s
graphics.drawobject d,100,100
```

Öffnen und Abspeichern von Vektorgrafiken

Zwei Methoden der FolderItem-Klasse sind für die Verwendung mit Vektorgrafiken von Bedeutung: OpenAsVectorPicture und SaveAsPicture. Die OpenAsVectorPicture-Methode öffnet eine Pict-Datei (Macintosh) oder eine .emf-Datei (Windows) und versucht, die Objekte in der Pict-Datei in für die Group2D-Klasse von REALbasic verwertbare Objekte umzuwandeln. Die Originaldatei enthält möglicherweise Elemente, für die es keine Entsprechung in REALbasic gibt. Die OpenAsVectorPicture-Methode versucht das bestmögliche Ergebnis für derlei Objekte zu erzielen.

Die SaveAsPicture-Methode besitzt einen optionalen Parameter, der das Format der abgespeicherten Datei festlegt. Möglich sind das Pict-Format (Macintosh) oder die .emf- und .bmp-Formate (Windows). Die Informationen der Vektorgrafik gehen dabei nicht verloren. Weitere Informationen finden Sie in der Sprachreferenz.

Der Umgang mit Farben

REALbasic behandelt Farben als Datentypen. Eine Farbe wird durch drei Farbwerte definiert. Eine Farbe kann spezifiziert werden, indem man eines der Farbmodelle RGB, HSV oder CMY verwendet. Um die Farbe festzulegen, verwenden Sie die jeweils relevanten drei Farb-Eigenschaften. Wenn Sie z.B. das RGB-Model benutzen, verwenden Sie die RGB-Funktion, um die Werte für die Eigenschaften Rot, Grün und Blau festzulegen. Die Anteile können einen Wert von 0 bis 255 annehmen. Einige Klassen haben Color-Eigenschaften. Beispielsweise ist die ForeColor-Eigenschaft der Graphics-Klasse eine solche Color-Eigenschaft.

Soll eine Farbe gespeichert werden, erzeugen Sie eine Eigenschaft oder Variable vom Typ Color und weisen dieser mit der RGB-, HSV- oder CMY-Funktion einen Wert zu. Im Beispiel wird eine neue Variable vom Typ Color angelegt und die Werte für die Farbe Weiß werden über die Rgb-Funktion zugewiesen:

```
Dim c as Color
c=Rgb(255,255,255)
```

Sie können auch auf direktem Wege und ohne die RGB-, HSV- oder CMY- Funktionen einen Farbwert zuweisen. Um einen Farbwert zu definieren, verwenden Sie das RGB-Model mit der folgenden Syntax:

```
&cRRGGBB
```

RR ist der Hexadezimal-Wert für Rot, GG der Hexadezimal-Wert von Grün und BB der Hexadezimal-Wert von Blau. Das folgende Beispiel entspricht dem vorangegangen Beispiel:

```
c=&cFFFFFF
```

FF ist der Hexadezimal-Wert für 255. Es gibt einen einfachen Weg, die Hexdezimal-Werte zu bestimmen. In der **Neue Konstante**-Dialogbox können Sie eine Konstante vom Typ Color definieren und mit Hilfe der integrierten Farbauswahl

(Color Picker) eine Farbe auswählen. Nach getroffener Farbauswahl bestimmt REALbasic automatisch den Hexadezimal-Wert und trägt ihn im Feld „Wert“ der Dialogbox ein.

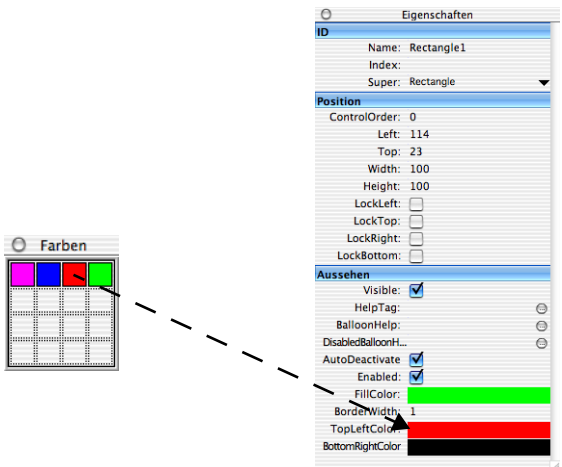
Es soll die ForeColor-Eigenschaft eines Graphics-Objekts auf Blau gesetzt werden, so dass der Text blau ausgegeben wird:

```
Sub Paint(g as Graphics)
 g.ForeColor=Rgb(9,13,80)
 g.DrawString "Hello World",50,50
End Sub
```

In der Entwicklungsumgebung können Sie Farben mit dem Farben- und Eigenschaftenfenster erzeugen und zuweisen. Das Farbenfenster besteht aus einer Palette von bis zu 16 Farben. Wenn Sie das Farbenfenster das erste Mal aufrufen, sind noch keine Farben zugewiesen. Klicken Sie auf ein leeres Kästchen des Farbenfensters, um ihm eine Farbe zuzuweisen. Es erscheint die Farbauswahl, in der Sie die gewünschte Farbe auswählen können. Ein Vorteil dieser Methode ist, dass Sie dazu keine RGB-, HSV- oder CMY-Werte kennen müssen.

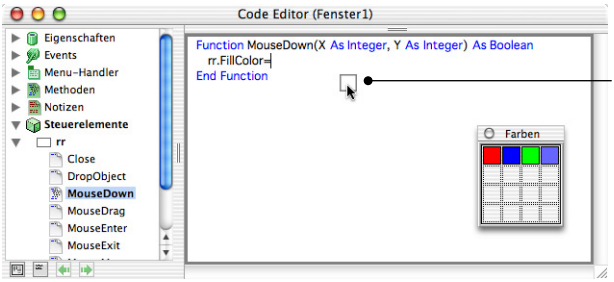
Nachdem Sie dem Farbenfenster Farben zugewiesen haben, können Sie einer Objekt-Eigenschaft, die eine Farbe akzeptiert, eine Farbe einfach durch das Draggen der Farbe aus dem Farbenfenster auf die gewünschte Eigenschaft im Eigenschaftenfenster zuweisen.

Abb. 211: Draggen einer Farbe aus der Farbpalette auf die gewünschte Eigenschaft im Eigenschaftenfenster



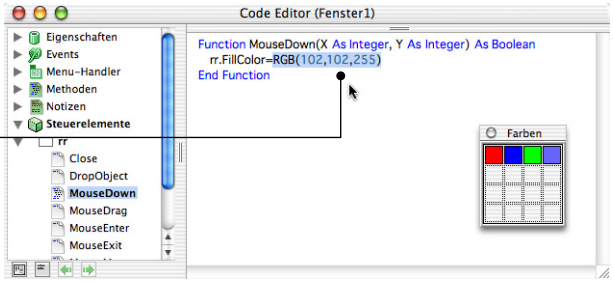
Wenn Sie mit dem Code-Editor arbeiten, können Sie den Wert einer Farbe des Farbenfensters einer Eigenschaft, die einen Farbwert annehmen kann, im Code zuweisen, indem Sie die Farbe in die entsprechende Codezeile draggen. Dies wird am besten anhand eines Beispiels verdeutlicht:

Abb. 212: Durch Draggen aus dem Farbenfenster einer Eigenschaft einen Farbwert zuweisen



Der Benutzer zieht mit der Maus eine Farbe aus der Farbpalette zur Einfügemarke im Code-Editor.

Wenn er den Mausknopf los lässt, erscheint an der Einfügemarke der Code für die entsprechende Farbe.



In der Abbildung hat der Benutzer zuerst die Einfügemarke hinter das Gleichheitszeichen einer Codezeile gesetzt, die einer Eigenschaft einen Farbwert zuweist. Anschließend draggt er eine Farbe aus dem Farbenfenster in diese Codezeile. Wenn er die Maustaste loslässt, wird der RGB-Wert für diese Farbe an der Einfügemarke eingesetzt. Da eine gedraggte Farbe den Farbwert auch als Text enthält, kann eine Farbe aus dem Farbenfenster auch auf ein anderes Objekt gedraggt werden, das gedraggten Text annimmt , wie beispielweise der Finder oder ein Texteditor wie BBEdit.

Ermitteln des RGB-Werts einer Farbe

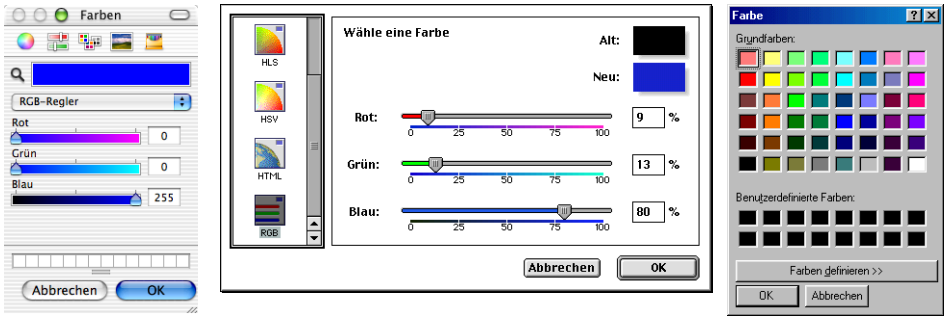
Wenn Sie zur Laufzeit eine Farbe zuweisen müssen und nicht wissen, welche RGB-Werte zu einer bestimmten Farbe gehören, können Sie den Mac OS-Color Picker verwenden, um diese Werte zu ermitteln. Folgender Code zeigt den Mac OS-Color-Picker:

```
Dim c as Color
Dim b as Boolean
b=SelectColor(c,"Wähle eine Farbe")
if b then //Anwender hat eine Farbe gewählt
// Mache hier irgendetwas mit dieser Farbe
end if
```

Wenn der Anwender die Color-Picker-Dialogbox mit „Abbrechen“ verlassen hat, hat die Variable „b“ den Wert False, andernfalls wird die ausgewählte Farbe im Color-Objekt „c“ zurückgeliefert und steht für die Zuweisung zur Farbeigenschaft eines Objektes zur Verfügung.

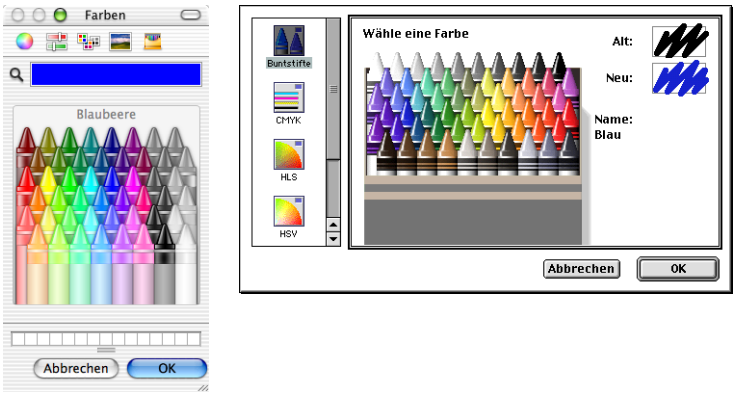
Sie haben diesen vielleicht schon einmal eingesetzt, um einem Steuerelement im Interface-Builder eine Farbe zu geben. Im Classic Mac OS und unter Windows sieht er aus wie in der folgenden Abbildung.

Abb. 213: Die Macintosh- und Windows-Farbauswahl



Die Mac OS 8/9-Farbauswahl bietet mehrere Farbmodelle. Bei der Farbauswahl von Mac OS X klicken Sie auf eine Farbe und es erscheint eine Vorschau im Farbfeld am oberen Fensterrand. Sobald Sie auf **OK** klicken, wandelt REALbasic Ihre Auswahl in RGB-Werte um. Wenn Sie eine leichter zu überschauende Auswahl an Farben wünschen, dann wählen Sie den Crayon Picker (Farbstift-Auswahl). Den gibt es unter Mac OS X und Mac OS 9.

Abb. 214: Die Crayon- (Buntstift-) Farbauswahl unter Mac OS



Der Crayon Color Picker zeigt Stifte mit den gebräuchlicher Farben. Um eine Farbe auszuwählen, klicken Sie einen Stift an. Die Windows-Version des Color Pickers kennt nur eine Darstellung. Sie können entweder eine der vorgegebenen Farben auswählen (so wie Sie es beim Apple Crayon Picker tun würden) oder Sie klicken auf „Farben definieren“, um sich den umfangreicheren Color Picker anzeigen zu lassen. Hier können Sie die Farbe über das RGB- oder das HSV-Model festlegen.

Abb. 215: Farbauswahl unter Windows XP

Wählen Sie eine Farbe aus, indem Sie auf einen Punkt im Farbspektrum klicken oder geben Sie bei RGB bzw. HSV die entsprechenden Werte ein. Klicken Sie auf „Farben hinzufügen“, um die Farbe in die Gruppe der anderen Farbbeispiele auf der linken Seite dieser Dialogbox mitaufzunehmen.

Die Pixel-Eigenschaft von Graphics-Objekten

Die Pixel-Eigenschaft eines Graphics-Objekts erlaubt das Setzen und das Auslesen eines beliebigen Pixels. Die Eigenschaft hat den Typ Color. Im Beispiel wird im Paint-Event-Handler ein Pixel schwarz eingefärbt, wenn es weiß ist und weiß eingefärbt, wenn es schwarz ist:

```
Sub Paint(g As Graphics)
 If g.Pixel(10,20)=Rgb(0,0,0) Then
 g.Pixel(10,20)=Rgb(255,255,255)
 Else
 g.Pixel(10,20)=Rgb(0,0,0)
  End If
End Sub
```

Man sieht, dass der Code für das Lesen der Farbe dem Code zum Setzen der Farbe entspricht.

Drucken von Text und Grafik

REALbasic erlaubt Ihnen, die Papierformat-Dialogbox aufzurufen und die Einstellungen des Benutzers zu sichern oder die Druckdialogbox vor dem Ausdruck anzuzeigen. Sie haben die Wahl, ob Sie vor dem Drucken die Drucken-Dialogbox anzeigen oder nicht.

Beim Drucken geht man fast genauso wie beim Ausgeben von Text oder Grafik in einem Canvas-Steuerelement oder in einem Fenster vor. Über die Funktionen OpenPrinter oder OpenPrinterDialog erhält man ein Graphics-Objekt. Ruft man die NextPage-Methode dieses Objekts auf, wird die Seite ausgegeben. Sie gibt das Graphics-Objekt an den Drucker weiter und löscht seinen Inhalt, so dass Sie mit dem Aufbau der nächsten Seite beginnen können.

Die Papierformat-Dialogbox

Über die Klasse PrinterSetup kann ein Objekt erzeugt werden, in dem man die Papierformat-Dialogbox anzeigen kann. Um diese anzuzeigen, rufen Sie die **PageSetupDialog**-Methode des PrinterSetup-Objekts auf. Diese Methode liefert den Wert True, wenn der Benutzer den OK-Knopf im Dialog geklickt hat oder False, wenn der Abbruch-Knopf geklickt wurde. Die PrinterSetup-Klasse bietet Eigenschaften für alle Einstellungen in der Papierformat-Dialogbox (Seitenausrichtung, Skalierung und so weiter). Eine Zusammenfassung der Eigenschaften finden Sie unter Printer-Klasse im Sprachreferenz-Handbuch. Meistens werden Sie allerdings mit den einzelnen Eigenschaften nichts zu tu haben, da sie allesamt in der SetupString-Eigenschaft zusammengefasst sind. Die SetupString-Eigenschaft kann gelesen und geschrieben werden, so dass Sie damit sehr einfach alle Einstellungen des PrinterSetup speichern und später wiederherstellen können. In einem Programm, mit dem der Benutzer Dokumente bearbeiten kann, könnte man dem Dokumentfenster eine Eigenschaft vom Typ String geben, in der man den Wert des SetupString speichern kann. Wählt der Benutzer den Menüpunkt **Seitenformat**, der sich bei den meisten Programmen im **Ablage**-Menü befindet, wird ein PrinterSetup-Objekt erzeugt, dessen SetupString-Eigenschaft alle Einstellungen enthält. Danach wird die Seitenformat-Dialogbox dargestellt und zeigt diese Einstellungen an. Im Beispiel heißt die Eigenschaft „Settings“:

```
Dim ps as PrinterSetup
ps=New PrinterSetup
ps.SetupString=Settings
If ps.PageSetupDialog Then
 Settings=ps.SetupString
End if
```

Klickt der Benutzer in der Seitenformat-Dialogbox auf OK, wird der Eigenschaft Settings der Wert des SetupStrings zugewiesen, da der Benutzer möglicherweise neue Einstellungen vorgenommen hat.

Den OpenPrinter- und OpenPrinterDialog-Funktionen kann man optional PrinterSetup-Objekte als Parameter übergeben, so dass die Einstellungen beim Ausdruck verwendet werden können.

Sollen die Druckeinstellungen mit dem Dokument gespeichert werden, dann werden Sie diese vermutlich in der Resource Fork des Dokuments ablegen wollen. Mehr dazu erfahren Sie im Kapitel „Mit Dateien arbeiten“ auf Seite 280.

Die Papierformat-Dialogbox wird momentan unter Linux nicht unterstützt. Der Aufruf der PrinterSetup-Funktion liefert False zurück und es wird kein Dialog angezeigt.

Ausdrucken mit der Druckdialogbox

Um die Druckdialogbox darzustellen und den Ausdruck zu starten, verwenden Sie die Funktion OpenPrinterDialog. Klickt der Benutzer in der Dialogbox auf OK, dann liefert die Funktion ein Graphics-Objekt. Klickt er auf Abbruch, gibt die Funktion Nil zurück. Für den Aufbau der ersten Druckseite verwenden Sie das Graphics-Objekt, das Ihnen die Funktion geliefert hat und zeichnen in diesem Objekt mittels der üblichen grafischen Methoden wie DrawString, DrawLine, DrawOval, DrawPicture, etc. Ist die Seite komplett, wird sie mit der NextPage-Methode zum Drucker geschickt, die das Graphics-Objekt danach wieder leert, so dass Sie mit dem Aufbau der nächsten Seite beginnen können.

Dieses Beispiel gibt „Hello“ auf der ersten und „World“ auf der zweiten Seite aus:

```
Dim page as Graphics
page=OpenPrinterDialog()
If page<> nil Then
 page.DrawString "Hello", 50, 50
 page.NextPage
 page.DrawString "World", 50, 50
 page.NextPage
End
```

Die Angaben über die zu druckenden Seiten, die der Benutzer in der Dialogbox festlegen kann, werden automatisch berücksichtigt. Hat der Benutzer „von Seite 2 bis Seite 2“ eingestellt, wird nur die zweite Seite ausgedruckt.

Haben Sie die SetupString-Eigenschaft eines PrinterSetup-Objekts gespeichert, können Sie diese optional an die OpenPrinterDialog-Funktion weitergeben, damit diese Einstellungen berücksichtigt werden. Im Beispiel wird angenommen, dass der SetupString in einer Eigenschaft namens „Settings“ abgelegt ist. Die Einstellungen werden dann an den OpenPrinterDialog weitergegeben:

```
Dim page as Graphics
Dim ps as PrinterSetup
ps=New PrinterSetup
If Settings <> "" Then
 ps.SetupString=Settings
End If
page=OpenPrinterDialog(ps)
If page <> nil Then
 page.DrawString "Hello", 50, 50
 page.NextPage
 page.DrawString "World", 50, 50
 page.NextPage
End If
```

Weitere Informationen zur OpenPrinterDialog-Funktion finden Sie in der Sprachreferenz.

Drucken ohne Druckdialog

Um direkt zu drucken, ohne zuvor die Druckdialogbox zu zeigen, rufen Sie die **OpenPrinter**-Funktion auf. Die Funktion arbeitet analog zur OpenPrintDialog-Funktion. Mehr dazu finden Sie unter OpenPrinter im Sprachreferenz-Handbuch.

Drucken von Styled Text

EditFields sind in der Lage, Texte in verschiedenen Schriftarten und -größen darzustellen. Für den Ausdruck solcher Texte ist die Klasse StyledTextPrinter zuständig, die die Methode DrawBlock (anstelle von DrawString) verwendet. Es folgt ein einfaches Beispiel, das den Inhalt eines EditFields als Styled Text ausdruckt (Die StyledTextPrinter-Methode liefert Nil, wenn die MultiLine-Eigenschaft des EditFields nicht auf True gesetzt ist. Natürlich kann man auch keinen Styled Text in ein EditField eingeben, wenn die Eigenschaften MultiLine und Styled nicht True sind).

```
dim stp as StyledTextPrinter
dim g as Graphics
g=openPrinterDialog()
if g <> nil then
  stp=EditField1.StyledTextPrinter(g,72*7.5)
  stp.drawBlock 0,0,72*9
end if
```

DrawBlock erwartet als Parameter die X,Y-Koordinate, an der die linke obere Ecke des Blocks auf der Seite erscheinen soll, und die Höhe des Blocks. Das Beispiel gibt den Block links oben auf der Seite aus. Weitere Informationen finden Sie in der Sprachreferenz unter StyledTextPrinter.

Das REALbasic-Tutorial enthält ein Kapitel, wie man Styled Text in einem EditField druckt. Es enthält Code zum Drucken von mehr als einer Seite und zum Einstellen des druckbaren Bereichs auf der Seite. Weitere Informationen hierzu finden Sie im Tutorial ab Seite 50.

Text und Grafik über das Clipboard austauschen

Das Clipboard ist eine Klasse. Die Methoden und Eigenschaften des Clipboard erlauben Ihnen festzustellen, welche Daten sich im Clipboard befinden, Daten an das Clipboard zu senden und Daten vom Clipboard zu empfangen. Das Clipboard kennt drei Arten von Daten: Text, Grafik und Binärdaten. Letztere werden in einem String gespeichert und mit einem Datentyp Ihrer Wahl markiert, so dass man feststellen kann, welche Art von Daten im Clipboard vorliegen.

Bei EditFields kümmert sich REALbasic automatisch um die Behandlung von Cut, Copy und Paste im Edit-Menü. Bei anderen Steuerelementen, die Daten enthalten können, wie Canvas oder ListBox, ist dies aber nicht der Fall.

Um auf das Clipboard zuzugreifen, müssen Sie zunächst ein Objekt vom Typ Clipboard erzeugen:

```
Dim c as Clipboard
c=New Clipboard
```

Im Event-Handler, der das Clipboard öffnete, müssen Sie die Close-Methode des Clipboard-Objekts aufrufen, damit kein Fehler auftritt.

Feststellen des Typs der Daten im Clipboard

Sie können den Typ der Daten im Clipboard mit folgenden Methoden und Eigenschaften feststellen, die jeweils True oder False zurückliefern: TextAvailable, PictureAvailable und MacDataAvailable. MacDataAvailable wird verwendet, um herauszufinden, ob eine bestimmte Art von Binärdaten im Clipboard liegt (gewöhnlich testen Sie damit auf Daten, die Ihre eigene Anwendung dort abgelegt haben könnte). Um die MacDataAvailable-Methode zu verwenden, müssen Sie ihr den MacType-String übergeben, mit dem der entsprechende Datentyp markiert ist. Der String wurde beim Ablegen der Daten auf dem Clipboard an das Clipboard übergeben.

Auslesen von Daten des Clipboards

Nachdem Sie wissen, welche Art von Daten auf dem Clipboard vorliegt, können Sie diese mit den Eigenschaften Text, Picture und MacData auslesen. Im Beispiel wird ein Text (sofern vorhanden) in die Variable „Cliptext“ geholt:

```
Dim c as Clipboard
Dim ClipText as String
c=New Clipboard
If c.TextAvailable Then
 ClipText=c.Text
End If
C.Close
```

Ist ein Bild im Clipboard, dann legen wir es in „ClipPict“:

```
Dim c as Clipboard
Dim ClipPict as Picture
c=New Clipboard
If c.PictureAvailable Then
 ClipPict=c.Picture
End If
C.Close
```

In nächsten Beispiel wurden Zeilen einer ListBox ins Clipboard kopiert. Diese werden nun ausgelesen und in eine ListBox eingefügt.

```
dim theRows as string
dim c as clipboard
c=New Clipboard
If c.MacDataAvailable("rows") Then
 theRows=c.MacData("rows")
 Do
 Listbox1.AddRow Left(theRows,InStr(theRows,Chr(13))-1)
 theRows=Mid(theRows,InStr(theRows,Chr(13))+1)
 Loop until theRows=""
End If
C.Close
```

Denken Sie daran, dass Sie die Close-Methode des Clipboard-Objekts im Event-Handler, der das Objekt geöffnet hat, aufrufen müssen, weil sonst ein Fehler auftritt.

Daten auf dem Clipboard ablegen

Sie können Text, Bilder oder Binärdaten (in Form eines Strings) im Clipboard ablegen. Dazu erzeugen Sie ein Clipboard-Objekt und verwenden die Methoden oder Eigenschaften, die zum Typ der Daten passen, die Sie im Clipboard ablegen möchten.

Tabelle 31. Methoden und Eigenschaften, um Daten im Clipboard abzulegen

Datentyp	Methode oder Eigenschaft
Text	SetText-Methode
Bild	Picture-Eigenschaft
Binärdaten	AddMacData-Methode

So wird Text ins Clipboard befördert:

```
Dim c as Clipboard
c=New Clipboard
c.SetText "Hello World"
```

```
c.Close
```

So kopiert man ein Bild vom Objekt Canvas1 in das Clipboard:

```
Dim c as Clipboard
c=New Clipboard
c.Picture=Canvas1.Backdrop
c.Close
```

Im nächsten Beispiel werden Zeilen einer ListBox in das Clipboard kopiert. Sie werden mit der AddMacData-Methode kopiert, so dass sie nicht als Text im Clipboard erscheinen:

```
Dim i as Integer
Dim c as Clipboard
Dim rows as String
c=New Clipboard
For i=0 to ListCount
 If Listbox1.Selected(i) Then
 rows=rows+Listbox1.List(i)+Chr(13)
 End If
Next
c.AddMacData rows,"rows"
c.Close
```

Denken Sie daran, dass Sie die Close-Methode des Clipboard-Objekts im Event-Handler, der das Objekt geöffnet hat, aufrufen müssen, weil sonst ein Fehler auftritt.

Animationen mit Sprites

Das SpriteSurface-Steuerelement wird benutzt, wenn Bilder bewegt werden sollen. Dabei kümmert sich SpriteSurface selbständig um das Neuzeichnen. Jedes Bild ist ein Sprite-Objekt. Diese haben Eigenschaften für die Koordinaten, an denen sie sich während der Animation befinden.

Bewegen von Sprites und Ändern des Bildes

Der NextFrame-Event-Handler wird immer dann aufgerufen, wenn SpriteSurface bereit ist, das nächste Bild (Frame) der Animation zu zeichnen. Wenn ein Sprite im nächsten Bild eine andere Position haben soll, ändern sie dessen X/Y-Eigenschaft im NextFrame-Event-Handler. Soll er sein Aussehen ändern, dann modifizieren Sie seine Image-Eigenschaft. Um einen Sprite aus der Animation zu löschen, rufen Sie dessen **Close**-Methode auf.

Neuzeichnen der Frames

Die Geschwindigkeit, mit der die Frames neu gezeichnet werden, hängt von der Eigenschaft FrameSpeed ab. Sie bestimmt, wie oft pro Sekunde das Monitor-Bild neu aufgebaut und der NextFrame-Event-Handler aufgerufen wird.

FrameSpeed legt fest, wie oft der Bildschirminhalt neu gezeichnet werden soll, bevor ein Frame neu gezeichnet wird. Eine FrameSpeed von 0 sorgt dafür, dass der Rechner die Frames mit der höchstmöglichen Geschwindigkeit neu zeichnet. Um die passende FrameSpeed zu berechnen, teilen Sie 60 durch die Anzahl der Frames, die pro Sekunde gezeichnet werden sollen und runden Sie das Ergebnis auf den nächsten Integer-Wert. Jedes Neuzeichnen eines Frames löst ein NextFrame-Event aus.

Beginn und Ende der Animation

Um eine Animation zu starten, rufen Sie die **Run**- oder **Update**-Methode von SpriteSurface auf. Die Update-Methode lässt die Animation genau einen Schritt ausführen – sie zeichnet die Sprites neu, die sich bewegt oder geändert haben,

sendet Collisions-Events und ruft den NextFrame-Event genau einmal auf. Sie können das SpriteSurface animieren, indem Sie Update wiederholt aufrufen – z.B. über ein Timer-Steuerelement. Um die Animation zu stoppen, rufen Sie die **Stop**-Methode des SpriteSurface auf. Mit der **Close**-Methode übergeben Sie den Bildschirm an das Fenster, das das SpriteSurface-Steuerelement enthält.

Der SpriteSurface-Bereich

Das SpriteSurface ist ein „reguläres" Steuerelement mit den Eigenschaften Left, Top, Width und Height, die gelesen und geschrieben werden können. Die Größe des SpriteSurface-Steuerelements kann in der Entwicklungsumgebung geändert werden, indem man die Greifpunkte mit der Maus bewegt oder indem man diese Eigenschaften bearbeitet. Die Größe kann auch zur Laufzeit über diese Eigenschaften geändert werden.

Um ein SpriteSurface-Steuerelement so zu ändern, dass es den gesamten Bildschirm übernimmt, setzen Sie die FullScreen-Eigenschaft des Elternfensters auf True und MenuBarVisible auf False und schalten Sie die „Lock"-Eigenschaften ein, damit sich die Größe des SpriteSurface-Steuerelements zusammen mit dem Elternfenster ändert. Dies passt automatisch die Eigenschaften Left, Top, Width und Height an.

In der Backdrop-Eigenschaft von SpriteSurface kann ein Bild abgelegt werden, das bei Beginn der Animation angezeigt wird. Sie können dieses auch verändern, während die Animation läuft.

Sie sollten es jedoch vermeiden, in dem Bereich zu zeichnen, in dem gerade eine Animation abläuft. Dieser Bereich wird durch die Eigenschaften SurfaceWidth, SurfaceHeight, SurfaceLeft und SurfaceTop festgelegt.

Reagieren auf Benutzereingaben während der Animation

Im NextFrame-Event-Handler können Sie mit der KeyTest-Methode eventuell gedrückte Tasten abfragen. Sie übergeben der Methode den Key-Code einer Taste und KeyTest liefert True zurück, wenn die Taste gedrückt wurde. Key-Codes sind keine ASCII-Codes, weil einige Tasten keinen zugeordneten ASCII-Wert haben (wie beispielsweise Shift, ctrl oder Befehlstaste). Key-Codes sind spezielle Codes, die jeder Taste auf der Tastatur zugeordnet sind. Diese können allerdings von Tastatur zu Tastatur verschieden sein. Beispielsweise gibt es Unterschiede zwischen einer englischen und einer französischen Tastatur.

Abb. 216: Key-Codes auf verschiedenen Tastaturen

3D-Grafik-Animationen mit dem RB3DSpace-Steuerelement

Sie können das RB3DSpace-Steuerelement zum Erzeugen von Animationen im dreidimensionalen Raum verwenden. Es arbeitet in Verbindung mit 3D-Klassen, mit denen Sie 3D-Objekte laden, gruppieren und manipulieren können. Das Basis-3D-Objekt ist von der Object3D-Klasse abgeleitet; 3D-Objekte werden dem Anwender über das RB3DSpace-Steuerelement präsentiert.

Die 3D-Fähigkeiten von REALbasic basieren auf bestehenden 3D-Bibliotheken, die viel Power für einen geringen Preis bieten. Wenn Sie oder Ihre Endbenutzer Anwendungen benutzen, die 3D-Animationen verwenden, müssen die entsprechenden Bibliotheken installiert sein.

Zwei voneinander unabhängige Bibliotheken sind erhältlich: QuickDraw 3D von Apple, und Quesa, eine Open-Source-Bibliothek, die für verschiedene Plattformen erhältlich ist.

Unter Mac OS Classic benötigen Sie QuickDraw 3D, das standardmäßig installiert ist. Wenn es aus irgendwelchen Gründen nicht mehr vorhanden sein sollte, dann können Sie es über den QuickTime Installer neuinstallieren.

Unter Mac OS X müssen Sie Quesa und OpenGL installieren.

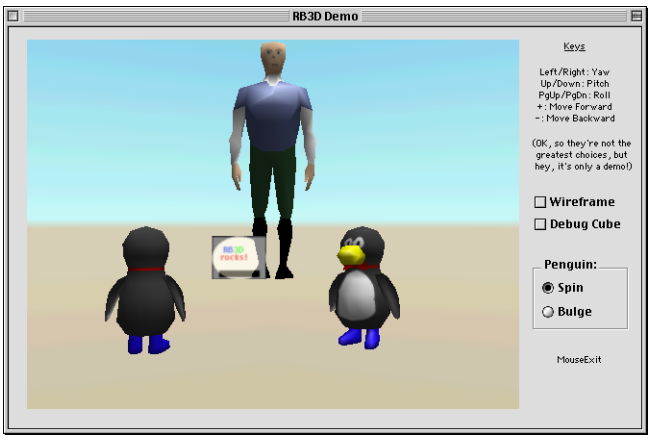
Unter Windows müssen Sie Quesa und OpenGL installieren.

Quesa erhalten Sie unter www.quesa.org. QuickTime erhalten Sie von Apple Computer unter www.apple.com/quicktime.

Da es sehr schwierig ist, 3D-Animationen auf einer gedruckten Seite zu illustrieren, sollten Sie sich am besten die 3D-Beispiel-Projekte von der REALbasic CD anschauen. Das RB 3D Demo illustriert die folgenden Bewegungen:

- Yaw: Cursor links/rechts
- Pitch: Cursor hoch/runter
- Zoom: +/- Tasten bewegen die „Kamera“ im 3D-Raum.

Der Hintergrund und jede Figur in der Demo ist ein 3DMF-Objekt, das in das RB3DSpace geladen wird.



Animationen werden mit den Tasten durchgeführt, die im rechten Teil des Fensters beschrieben sind und der zugehörige Code sieht folgendermaßen aus:

```
Select case asc(Key)
 case 28 // left
 View3D1.Camera.Yaw 0.1
 case 29 // right
 View3D1.Camera.Yaw -0.1
 case 30 // up
```

```
View3D1.Camera.Pitch -0.1
case 31 // down
View3D1.Camera.Pitch 0.1
case 11 // page up
View3D1.Camera.Roll -0.1
case 12 // page down
View3D1.Camera.Roll 0.1
case 43 // keypad +
View3D1.Camera.MoveForward 10.0
case 45 // keypad -
View3D1.Camera.MoveForward -10.0
case asc("r")
mPenguin.Yaw -0.1
else
Status.text = "Unbekannte Taste: " + str(asc(Key))
return true
end select


View3D1.Refresh
```

View3D1 ist das RB3DSpace Steuerelement und die Methoden Yaw, Pitch, Roll und MoveForward der Object3D-Klasse liefern die Animationen, wenn der Anwender die entsprechenden Tasten drückt.

Informationen darüber, wie Sie Objekte in eine 3D-Umgebung einbauen und animieren können, finden Sie in der Sprachreferenz.

Kapitel 8 Mit Dateien arbeiten

Die meisten Anwendungen lesen und schreiben Dateien. Einige sichern Daten in einem eigenen Format. Oft wird der Benutzer eine Dialogbox zum Laden oder Speichern der Dateien benötigen. REALbasic stellt dafür zahlreiche Funktionen zur Verfügung.

Inhalt

- Welche Dateitypen gibt es?
- Was sind FolderItems?
- Zugriff auf Dateien
- Text- und Binärdateien
- Bild-, Sound- und QuickTime-Dateien
- Lesen und Schreiben in der ResourceFork
- Umgang mit Dateien als Parameter

Welche Dateitypen gibt es?

Es gibt viele verschiedene Dateitypen. Der Dateityp definiert die Art der Daten, die in einer Datei abgelegt sind. Der Dateityp TEXT wird für Textinformationen benutzt, während PICT eine Bilddatei bezeichnet. Jede Datei, die unter Mac OS Classic gespeichert wird, hat einen Dateityp, der in vier Buchstaben codiert wird. Ferner wird davon unabhängig eine Kennung mit vier Buchstaben abgelegt, die aussagt, welches Programm die Datei erzeugt hat. Dies ist der Creator. Unter Windows oder Mac OS X verwendet man zur Unterscheidung der Dateitypen einen Suffix, den ein Benutzer auch noch beliebig verändern kann (ein mehr als zweifelhaftes Verfahren). Die Kennung des Datentyps macht es einem Programm leichter zu bestimmen, ob es mit dem Inhalt einer Datei etwas anfangen kann. Erwartet ein Programm Dateien, die einen Text enthalten, sollte es nur solche Dateien öffnen, die die Kennung TEXT tragen. Bei Bilddateien spricht man oft von PICT-Dateien, was darauf zurückzuführen ist, dass deren Kennung PICT ist. Programme haben die Kennung APPL, was dem Mac-Betriebssystem mitteilt, dass es sich um ein ausführbares Programm handelt. (Zum Vergleich: Unter Windows/DOS ist dies die Endung .exe, die man aber sehr einfach jeder beliebigen Datei verpassen kann).

Damit Sie sich nicht so intensiv damit beschäftigen müssen, verwaltet REALbasic diese Informationen über „file types". Ein solcher „file type" wird in Ihrem Projekt abgelegt und enthält die Kennungen für einen Creator (ihr Programm) und die Dateitypen. Jeder Dateityp hat einen Namen, den Sie in Ihrem Programm verwenden, wenn Sie Dateien öffnen oder speichern. So können Sie mit Begriffen arbeiten, die Sie selbst bestimmen können, statt sich kryptische Kennungen zu merken. Außerdem macht es das Projekt weniger abhängig vom Mac OS.

Die Creator-Codes unterscheiden kleine und große Buchstaben und müssen eindeutig einer Anwendung zugeordnet sein. Dazu kann man bei Apple einen Creator-Code registrieren. Dies geht im Internet unter:
http://developer.apple.com/dev/cftype/find.html.

Die Dialogbox „Dateitypen"

Die Dialogbox „Dateitypen" wird verwendet, um die Datentypen zu bestimmen, die Ihr Programm verstehen soll. Sie erreichen die Box unter Bearbeiten/Dateitypen.

Abb. 217: Die Dateitypen-Dialogbox

Diese Liste zeigt alle Dateitypen, die innerhalb des Projektes verwendet werden. Ohne weitere Angaben verwendet REALbasic zunächst den Dateityp „Text“ für alle Dateien. In der Dialogbox können weitere Typen angelegt werden.

Hinzufügen eines Dateityps

Der Dateityp „text“ für Dateien mit dem Type „TEXT“ ist bereits eingebaut. Das Dateityp-Vorlagen-Popup enthält ein paar nützliche Wildcard-Dateitypen. Sie sind in folgender Tabelle aufgelistet.

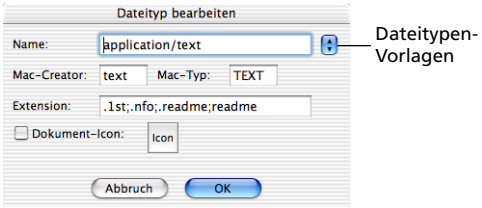
Tabelle 32. Spezielle Dateitypen, die im Dateitypen-Dialog eingebaut sind

Dateitypen-Name	Type	Creator	Beschreibung
special/any	*????*	*????*	Beliebiger Type und Creator
special/folder	*fold*	*MACS*	Beliebiger Ordner (nur Mac OS)
special/disk	*disk*	*MACS*	Beliebiges Volume (nur Mac OS)

Um einen neuen Dateityp zu bekommen:

1. Ist die Dialogbox „Dateityp“ nicht geöffnet, dann öffnen Sie diese mit **Bearbeiten/Dateitypen**.
2. Klicken Sie auf den **Neu**-Knopf.

Abb. 218: Dialogbox zum Hinzufügen von Dateitypen



3. In der Dialogbox wählen Sie aus den Vorgaben einen passenden Typ aus oder geben einen neuen Namen, Creator, Typ und eventuelle Suffixe an.
4. Danach sichern Sie die Änderungen mit einem Klick auf **OK**.

Ändern des Dateityps

1. Ist die Dialogbox „Dateityp“ nicht geöffnet, dann öffnen Sie diese mit **Bearbeiten/Dateitypen**.
2. Klicken Sie den gewünschten Typ an.
3. Klicken Sie auf den Knopf **Ändern**.
4. Tragen Sie Ihre Änderungen ein und klicken Sie auf **OK**.
5. Sichern Sie die Änderungen durch Verlassen der Dateityp-Dialogbox mit einem Klick auf **OK**.

Sollten Sie den Namen eines Dateityps ändern, müssen Sie alle Programmstellen, in denen dieser Name verwendet wurde, manuell ändern. Das ist mit der Suchen- und Ersetzen-Funktion leicht möglich.

Löschen eines Dateityps

1. Ist die Dialogbox „Dateityp“ nicht geöffnet, dann öffnen Sie diese mit **Bearbeiten/Dateitypen**.
2. Klicken Sie den gewünschten Typ an.
3. Klicken Sie auf den Knopf **Löschen**.
4. Sichern Sie die Änderungen durch Verlassen der Dateityp-Dialogbox mit einem Klick auf **OK**.

Wenn Sie einen Dateityp entfernen, sollten Sie sich sicher sein, dass dieser nirgends mehr im Projekt verwendet wird.

Dateitypen verwenden

Sie übergeben einen oder mehrere Dateitypen in einem Befehl, um zu erreichen, dass nur die übergebenen Dateitypen erlaubt sind. Folgendes Beispiel legt in einem Open-Event-Handler eines Steuerelements fest, dass es TEXT-Dateien akzeptiert, die vom Desktop aus darauf gezogen wurden.

```
me.AcceptFileDrop(“text”)
```

Der Befehl

```
f=GetOpenFolderItem("video/quicktime")
```

zeigt eine „Datei öffnen“-Dialogbox an, durch die der Anwender QuickTime-Filme sehen und öffnen kann. Um festzulegen, dass man mehr als einen Dateityp akzeptiert, listen Sie die Dateitypen durch Semikolon getrennt auf. Folgendes Beispiel erlaubt dem Anwender, Dateien vom Typ PICT, GIF und JPEG zu sehen und zu öffnen.

```
f=GetOpenFolderItem("image/x-pict;image/jpeg;image/gif")
```

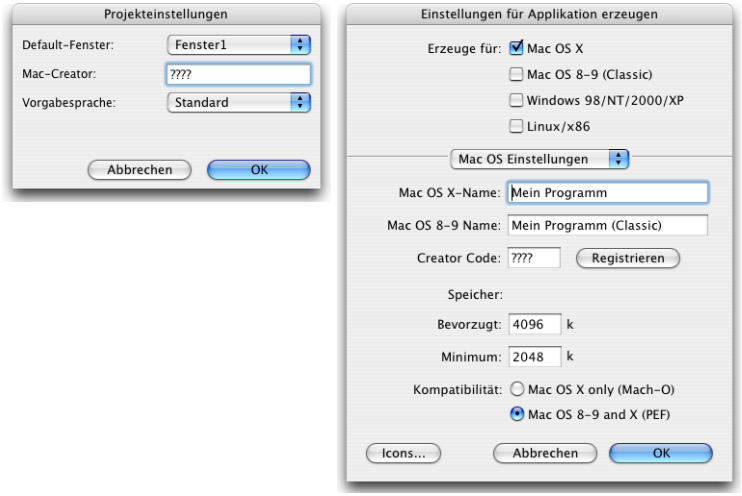
In der Sprachreferenz finden Sie zusätzliche Methoden, die Dateitypen als Argument besitzen.

Definieren eigener Icons für Ihr Programm

Die meisten Programme verwenden eigene Dateien und ordnen diesen auch Icons (Piktogramme) zu. Diese sehen normalerweise dem Icon des Programms ähnlich. Dadurch wird es einfacher, Dateien, die zu einem Programm gehören, zu identifizieren. Das klappt aber nur dann, wenn man dem Projekt einen Creator-Code gegeben hat und ein selbständig ablaufendes Programm erzeugt wurde (das unabhängig von der REALbasic-Entwicklungsumgebung läuft).

Um dem Projekt eigene Icons zuzuordnen, weisen Sie als erstes Ihrer Anwendung einen Creator-Code zu. Dies ist innerhalb der IDE in den Projekt-Einstellungen oder den Compiler-Einstellungen möglich.

Abb. 219: Creator-Code in den Projekt- oder Compilereinstellungen festlegen



Geben Sie Ihren Creator-Code entweder im Mac-Creator-Feld der Projekt-Einstellungen oder im Creator Code-Feld unter den Macintosh-Einstellungen in den Compiler-Einstellungen ein. Creator-Codes sind case-sensitiv.

1. Wählen Sie **Bearbeiten/Projekt-Einstellungen**.
2. Geben Sie eine Kennung für den Creator ein, die aus vier Buchstaben besteht.
3. Klicken Sie auf **OK**.
4. Wählen Sie **Bearbeiten/Dateitypen**.
5. Legen Sie einen neuen Dateityp an, oder ändern Sie einen bestehenden Typ.
6. Beachten Sie, dass der Creator der Datei genau mit dem unter den Projekteinstellungen festgelegten übereinstimmt.
7. Prüfen Sie, ob die Checkbox Datei-Icon angewählt ist.
8. Befördern Sie Ihr selbstdefiniertes Icon in das Clipboard.
9. Klicken Sie auf das vorgegebene Icon, so dass es selektiert wird.
10. Wählen Sie **Bearbeiten/Einfügen** (⌘-V oder Ctrl-V).
11. Klicken Sie auf **OK**.

Nachdem Sie das Piktogramm festgelegt haben, müssen Sie ein selbständig laufendes Programm erzeugen und danach die Desktop-Datenbank des Finders neu aufbauen (unter Mac OS Classic). Sonst wird das Icon noch nicht automatisch angezeigt. Die Desktop-Datenbank wird neu aufgebaut, wenn Sie den Computer neu starten und dabei die Befehlstaste (Apfel) und die Option-Taste (alt) gedrückt halten, bis der Mac Sie fragt, ob Sie die Desktop-Datenbank neu aufbauen möchten.

Was sind FolderItems?

REALbasic behandelt Laufwerke, Ordner, Anwendungen und Dateien als FolderItems. Schlicht gesagt ist alles ein FolderItem, was auf dem Desktop erscheinen kann. Das gilt für Dateien ebenso wie für den Papierkorb.

Die FolderItem-Klasse ist Ihr Zugang zu allem, was auf einem Speichermedium geschehen soll. Um eine Datei zu lesen, benötigen Sie ein FolderItem und die Methoden, die es zur Verfügung stellt. Es gibt viele verschiedene Methoden, ein FolderItem zu erhalten, das dann ein Verzeichnis (Volume), eine bestimmte Datei etc. repräsentiert. Der Benutzer kann

es über eine Dialogbox wählen, oder Sie können es aus einem bestimmten Pfad lesen oder es sogar von einem anderen FolderItem geliefert bekommen. FolderItems verfügen über Eigenschaften, die den Pfad, den Namen, die Größe, den Typ und so weiter beinhalten. Außerdem bieten sie Methoden zum Anlegen, Löschen, Öffnen, Kopieren etc.

Detaillierte Informationen dazu unter FolderItem in der Sprachreferenz.

Wie werden Aliase benutzt?

Aliase sind Dateien, die als Stellvertreter für die eigentliche Datei agieren und weder genauso heißen müssen, noch an derselben Stelle gespeichert sein müssen wie die Datei, für die sie stehen. REALbasic stellt Befehle bereit, die es Ihnen erlauben, entweder mit dem Originalobjekt zu arbeiten oder mit dem Alias selbst zu hantieren.

Die Funktion GetFolderItem löst einen Alias automatisch auf, während GetTrueFolderItem mit dem Alias selbst arbeitet.

Lesen einer Datei an einer vorgegebenen Stelle

Wenn Sie den vollständigen Pfad zu einer Datei kennen, können Sie mit dessen Hilfe auf die Datei zugreifen. Wenn Sie aber beispielsweise auf das Dokument „Zeitplan“ zugreifen wollen, welches sich im gleichen Verzeichnis befindet wie die Applikation, so benötigen Sie einen relativen Pfad. Der relative Pfad beginnt in dem Verzeichnis, in dem sich die Applikation befindet. Deshalb brauchen Sie nur den Namen des Dokuments („Zeitplan“) anzugeben, da es sich im gleichen Verzeichnis befindet, wie die Applikation.

Der folgende Code erzeugt ein FolderItem-Objekt, das das Dokument „Zeitplan“ repräsentiert.

```
Dim f as FolderItem
f=GetFolderItem("Schedule")
```

Die GetFolderItem-Funktion sollte nur verwendet werden, wenn Sie eine Datei oder ein Verzeichnis im Applikations-Verzeichnis oder einen leeren String (““) angeben. Letzteres liefert ein FolderItem-Objekt für das Verzeichnis, in dem sich die Applikation befindet.

```
Dim f as FolderItem
f=GetFolderItem("") //liefert das Applikations-Verzeichnis
```

Der vollständige Pfad, auch absoluter Pfad genannt, beginnt mit dem Volume-Namen, gefolgt von einer Reihe Verzeichnisnamen, die mit einem Trennzeichen voneinander getrennt sind. Unter Macintosh ist dieses Trennzeichen ein Doppelpunkt, unter Windows ein Backslash. Das letzte Element des Pfades, ist der Dateiname oder Verzeichnisname, auf das Sie zugreifen möchten.

Um einen absoluten Pfad zu erzeugen, sollten Sie mit der Volume-Funktion beginnen, der Sie das Volume übergeben, auf dem sich die Datei oder das Verzeichnis befindet. Nun können Sie mit Hilfe der Child-Methode der FolderItem-Klasse zum gewünschten Verzeichnis bzw. zur gewünschten Datei navigieren. Die Volume-Funktion liefert ein FolderItem-Objekt für das angegebene Volume. Als Parameter geben Sie den Index des Volumes an, wobei der Index 0 immer das Boot-Volume liefert, auf dem sich das Betriebssystem befindet.

```
Dim f as FolderItem
f=Volume(0) // Liefert ein FolderItem für das Boot-Volume
```

Die Parent-Methode hingegen liefert das übergeordnete Verzeichnis zurück. Dies funktioniert natürlich nicht, wenn Sie das FolderItem eines Volumes angeben. Über die Child-Methode greifen Sie auf Einträge zu, die eine Verzeichnisebene tiefer liegen.

Sie können mit Hilfe der Child-Methode einen absoluten Pfad erzeugen. Wenn Sie auf die Datei „Zeitplan“ im Verzeichnis „Verschiedenes“ auf dem Boot-Volume zugreifen möchten, geben Sie folgendes an:

```
Dim f as FolderItem
f=Volume(0).Child("Verschiedenes").Child("Zeitplan")
```

Das folgende Beispiel verwendet den relativen Pfad, um mit Hilfe der Parent-Methode das FolderItem des Verzeichnisses zu bestimmen, in dem sich das Verzeichnis der Applikation befindet.

```
Dim f as Folderitem
f=GetFolderItem("").Parent
```

Haben Sie ein FolderItem für die gewünschte Datei oder das Verzeichnis, können Sie dieses – abhängig vom Typ – kopieren, löschen, umbenennen, lesen, schreiben etc. Weitere Informationen dazu finden Sie weiter hinten in diesem Kapitel.

Sicherstellen, dass man Zugriff auf das Objekt hat

Wenn Sie versuchen, ein FolderItem für eine Datei oder ein Verzeichnis zu ermitteln, können zwei Dinge schief gehen. Als erstes kann der Pfad ungültig sein. Ein ungültiger Pfad enthält einen Laufwerks- oder Ordnernamen, der nicht existiert. Wenn der Volume-Index 1 ist, der Benutzer aber nur ein Volume (0) besitzt, liefert die Volume-Funktion Nil als Wert für die FolderItem-Instanz f. Wenn Sie versuchen, eine Eigenschaft oder Methode der FolderItem-Klasse zu verwenden, wird eine NilObjectException auftreten. Wenn diese Exception nicht abgefangen wird, stürzt das Programm ab. Das sollte man verhindern.

Zweitens kann es sein, dass die Datei, auf die Sie zugreifen wollen, nicht existiert.

Folgender Code prüft diese beiden Situationen:

```
Dim f as FolderItem
f=Volume(0).Child("Dokumente").Child("Zeitplan")
If f <> Nil then
       If f.Exists then
              //Zugriff auf das FolderItem
       Else
              MsgBox "Das Dokument existiert nicht!"
       End if
Else
       MsgBox "Ungültiger Pfad angegeben!"
End if
```

Wenn der Pfad gültig ist, prüft der Code die Exists-Eigenschaft des FolderItems, um sich zu vergewissern, dass die Datei schon vorhanden ist. Ist die Datei nicht vorhanden oder der Pfad ungültig, wird eine Warnmeldung angezeigt.

Sie können einen ungültigen Pfad durch einen Exception Block abfangen. Mehr dazu erfahren Sie im Abschnitt „Runtime Exception Fehler“ auf Seite 360.

Auslesen von Informationen über ein FolderItem

Wenn GetFolderItem ein gültiges FolderItem auf ein vorhandenes Objekt liefert, können Sie mit der lokalen Variablen „f“ auf alle Informationen des FolderItems zugreifen. Zum Beispiel können Sie wie folgt das Änderungsdatum (modification date) herausbekommen:

```
Dim f as FolderItem
f=Volume(0).Child("Dokumente").Child("Zeitplan")
if f <> Nil then
       if f.Exists then
       MsgBox f.ModificationDate.ShortDate
       End if
End if
```

Löschen eines FolderItem

Für das Löschen ist die Methode „Delete“ des FolderItem zuständig:

```
Dim f as FolderItem
f=Volume(0).Child("Dokumente").Child("Zeitplan")
If f <> nil Then
       if f.Exists then
       f.Delete
       End if
End if
```

Ist das FolderItem geschützt, tritt dabei ein Fehler auf. Mit der Eigenschaft „Locked“ des FolderItems lässt sich das feststellen.

Achtung: Das Löschen eines FolderItems verschiebt es nicht in den Papierkorb. Es wird sofort dauerhaft gelöscht.

Auslesen des Pfadnamens der Anwendung

Das Übergeben einer leeren Zeichenkette (Null-String, zwei direkt aufeinander folgende Anführungszeichen) an die GetFolderItem-Funktion liefert ein FolderItem zurück, das den Pfad enthält, in dem das Programm/Projekt sich befindet. Mit den Methoden dieses FolderItems können Sie auf alle Dateien zugreifen, die sich in dem Ordner befinden, in dem Ihr Programm gespeichert ist. Die Item-Eigenschaft liefert ein Array von FolderItems mit allen Objekten des Ordners.

Zugriff auf Dateien im Pfad der Anwendung

Ist der erste Teil eines Pfadnamens nicht der Name eines Volumes, geht die GetFolderItem-Funktion davon aus, dass sich das entsprechende Objekt im selben Verzeichnis befindet, in dem auch Ihr Programm liegt. Wird ein Projekt innerhalb von REALbasic gestartet, dann ist dies eben das Verzeichnis, in dem das Projekt gespeichert ist. Sollte es noch nicht gespeichert worden sein, dann ist dies der Pfad, in dem REALbasic sich befindet.

Die Datei „Eigene Vorlagen“ im Ordner „Vorlagen“, der sich im Verzeichnis des Programms befindet, erreicht man so:

```
Dim f as FolderItem
f=GetFolderItem(":Vorlagen:Eigene Vorlagen")
```

Wenn die Datei sich im gleichen Ordner wie das Programm befindet, benötigen Sie den führenden Doppelpunkt nicht:

```
Dim f as FolderItem
f=GetFolderItem("Eigene Vorlagen")
```

Wenn sich die Datei im gleichen Ordner wie das Programm befindet, benötigen Sie den Doppelpunkt am Anfang nicht. Folgendes würde funktionieren:

```
Dim f as FolderItem
f=GetFolderItem("Eigene Vorlagen")
```

Zugriff auf System-Ordner

REALbasic hat spezielle Funktionen, um auf die wichtigen Systemverzeichnisse zuzugreifen. Die im folgenden aufgelisteten Funktionen werden auch dann korrekt arbeiten, wenn sich zwischenzeitlich der Name eines Ordners ändert sollte. Die Funktionen sind auch unabhängig von der Sprache und der Betriebssystemplattform. Mehr dazu finden Sie auch in der Sprachreferenz.

- AppleMenuFolder
- ControlPanelsFolder
- DesktopFolder
- ExtensionsFolder
- FontsFolder
- PreferencesFolder
- ShutDownItemsFolder

- StartupItemsFolder
- SystemFolder
- TemporaryFolder
- TrashFolder

So kann man die Anzahl der Dateien im Papierkorb herausfinden:

```
Dim f as FolderItem
f=TrashFolder
MsgBox "Objekte im Papierkorb: "+Str(f.Count)
```

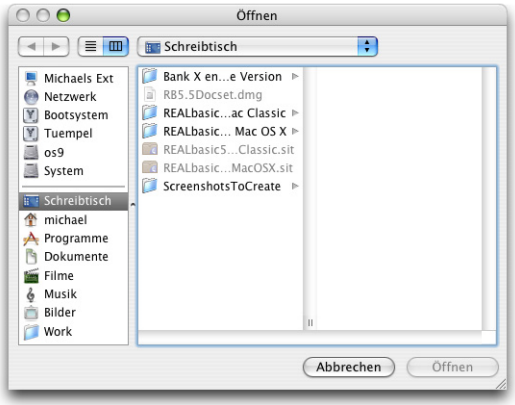
Lesen des Dateinamens aus einer „Datei öffnen“- Dialogbox

Die „Datei öffnen“-Dialogbox gestattet dem Benutzer das Auswählen einer Datei auf jedem verfügbaren Volume.

Wenn das Programm unter Mac OS 8.5 oder neuer läuft, wird die Dateiauswahlbox der Navigation Services angezeigt.

Abb. 220: Die Standarddateiauswahlbox ab Mac OS 8.5

Abb. 221: Der Mac OS X (10.3) Datei-Browser



Um dem Anwender eine Dialogbox zum Öffnen anzubieten, können Sie entweder die GetOpenFolderItem-Funktion aufrufen oder die OpenDialog-Klasse verwenden. Erstere ist eine Standard-Funktion, die den Standard-„Datei öffnen“-Dialog anzeigt. Die zweite erlaubt Ihnen, einen eigenen „Datei öffnen“-Dialog zu erzeugen, in dem Sie folgende Aspekte festlegen können:

- Position (Left- und Top-Eigenschaften)
- Ausgangsverzeichnis (Initial Directory-Eigenschaft)
- Gültige Dateitypen, die angezeigt werden sollen (Filter-Eigenschaft)

- Beschriftung der Bestätigen- und Abbrechen-Knöpfe (ActionButtonCaption- & CancelButtonCaption-Eigenschaften)
- Text, der im Titelbalken des Dialogs erscheint (Title-Eigenschaft)
- Text, der innerhalb des Dialogs erscheint (PromptText-Eigenschaft)

Die GetOpenFolderItem-Funktion zeigt eine „Datei öffnen"-Dialogbox an und liefert ein FolderItem-Objekt, das die Datei, die der Anwender auswählte, repräsentiert. Einer oder mehrere Dateitypen, die in der Dateitypen-Dialogbox für das Projekt definiert sein müssen, müssen dazu als Parameter übergeben werden. Nur Dateien dieser Typen werden dem Benutzer zur Auswahl angeboten. Mehrere Typen werden durch Semikolon getrennt.

Das folgende Beispiel zeigt eine „Datei öffnen"-Dialogbox, in der der Benutzer eine JPEG- oder PICT-Datei wählen kann. Danach wird das Änderungsdatum der selektierten Datei angezeigt:

```
Dim f as FolderItem
f=GetOpenFolderItem("image/jpeg;image/x-pict")
Msgbox f.ModificationDate.ShortDate
```

Klickt der Benutzer auf Abbruch statt auf Öffnen, liefert die Funktion „Nil" zurück. Auch hier müssen Sie wieder darauf achten, dass Sie das überprüfen, bevor Sie auf die Datei zugreifen, da sonst ein NilObjectException Fehler auftreten kann:

```
Dim f as FolderItem
f=GetOpenFolderItem("image/jpeg;image/x-pict")
If f <> Nil Then
       Msgbox f.ModificationDate.ShortDate
End if
```

Wenn Sie die OpenDialog-Klasse verwenden, erzeugen Sie ein neues Objekt, das auf dieser Klasse basiert und weisen seinen Eigenschaften Werte zu, um das Erscheinungsbild zu steuern. Folgendes Beispiel verwendet eine benutzerdefinierte Box und zeigt nur einen Dateityp an.

```
Dim dlg as OpenDialog
Dim f as FolderItem
dlg=New OpenDialog
dlg.InitialDirectory=GetFolderItem("HD:FrameMaker6.0:OnlineManuals")
dlg.Title="Wähle eine MIF-Datei"
dlg.Filter="application/x-mif"
f=dlg.ShowModal()
If f <> Nil then
 //Zeigt den vollständigen Pfad zu den Dateien in einem EditField an
 resultEdit.text = dlg.Result.AbsolutePath
Else
 resultEdit.text = "Benutzer-Abbruch"
End if
```

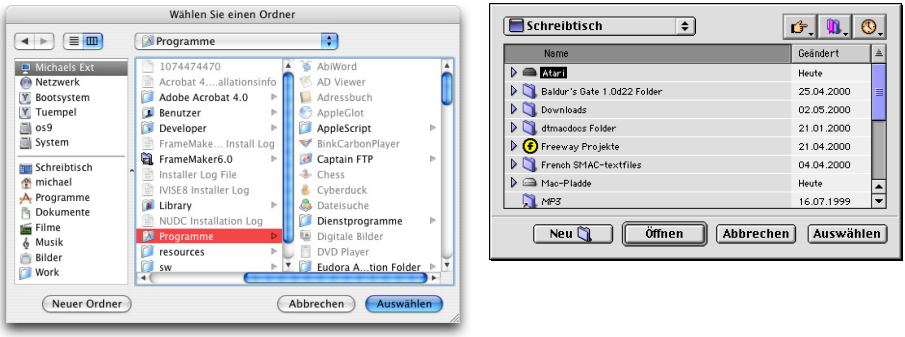
Mehr dazu siehe OpenDialog-Klasse und GetOpenFolderItem in der Sprachreferenz. Weitere Informationen zu Dateitypen siehe „Welche Dateitypen gibt es?" auf Seite 280.

Auslesen des gewählten Ordners

Die „Datei öffnen"-Dialogbox gestattet dem Benutzer nicht, einen Ordner auszuwählen. Daher gibt es in REALbasic die Funktion SelectFolder, die eine „Ordner wählen"-Dialogbox zeigt, in der der Benutzer einen Ordner wählen kann. Die SelectFolder-Dialog-Klasse bietet die gleiche Funktion, erlaubt aber das Erscheinungsbild des Dialogs zu ändern.

Wenn der Anwender Mac OS 8.5 oder neuer einsetzt, erscheint die neue Auswahlbox der Navigation Services:

Abb. 222: Die „Ordner auswählen“-Dialogbox unter Mac OS X und Windows



Die SelectFolder-Funktion liefert ein FolderItem, das den Ordner repräsentiert, den der Benutzer gewählt hat, wenn er den Knopf „Auswählen“ in der Dialogbox geklickt hat. Klickt er statt dessen auf „Abbrechen“ , liefert SelectFolder „Nil“. Auch das muss geprüft werden:

```
Dim f as FolderItem
f=SelectFolder
If f <> Nil Then
 MsgBox Str(f.Count)
End if
```

Wenn Sie die SelectFolderDialog-Klasse verwenden, erzeugen Sie ein Objekt, das auf dieser Klasse basiert und weisen seinen Eigenschaften Werte zu, um das Erscheinungsbild festzulegen. Sie können folgende Eigenschaften steuern:

- Position (Eigenschaften Left und Top)
- Ausgangsverzeichnis (Eigenschaft Initial Directory)
- Gültige Dateitypen, die angezeigt werden sollen (Eigenschaft Filter)
- Beschriftung der Bestätigen- und Abbrechen-Knöpfe (Eigenschaften ActionButtonCaption und CancelButtonCaption)
- Text, der im Titelbalken des Dialogs erscheint (Eigenschaft Title)
- Text, der innerhalb des Dialogs erscheint (Eigenschaft PromptText)

Folgender Code öffnet eine benutzerdefinierte „Ordner wählen“-Dialogbox und zeigt den Inhalt des „Dokumente“-Ordners auf dem Start-Volume im Browser an.

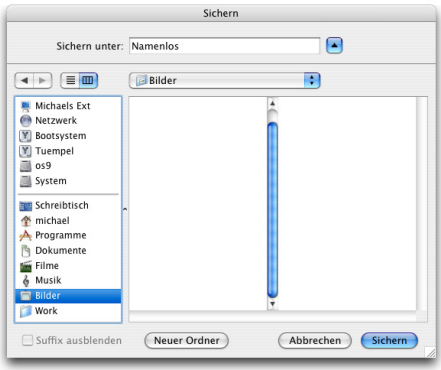
```
Dim dlg as SelectFolderDialog
Dim f as FolderItem
dlg=New SelectFolderDialog
dlg.ActionButtonCaption="Wähle"
dlg.InitialDirectory=Volume(0).Child("Dokumente")
f=dlg.ShowModal()
If f <> Nil then
 //Hier das FolderItem verwenden
else
 //Benutzer-Abbruch
end if
```

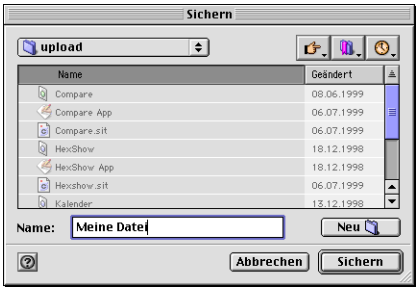
Mehr dazu siehe SelectFolder und SelectFolderDialog Klasse in der Sprachreferenz.

Benutzung der „Sichern unter“-Dialogbox

Die „Sichern unter“-Dialogbox (Save as) wird verwendet, um eine Datei unter einem neuen Namen zu sichern. Wenn der Anwender Mac OS 8.5 oder neuer einsetzt, erscheint die neue Auswahlbox der Navigation Services.

Abb. 223: Eine Sichern unter/Sichern als-Dialogbox





Die Funktion GetSaveFolderItem zeigt diese Dialogbox an. Die SaveAsDialog Klasse erlaubt Ihnen, eine benutzerdefinierte Version dieser Dialogbox zu erzeugen. Beide Objekte liefern ein FolderItem, das die Datei repräsentiert, die der Benutzer sichern möchte. Die Datei existiert zu diesem Zeitpunkt noch nicht. Sie werden später sehen, wie man den Programmcode realisiert, der die Datei anlegt und dann Daten in die Datei schreibt.

Beim Aufruf der GetSaveFolderItem-Funktion definieren Sie den Dateityp und einen Namensvorschlag für die Datei (dieser erscheint dann im Name-Feld in der „Sichern"-Dialogbox). Der Dateityp muss in der Liste der Dateitypen des Projekts definiert worden sein. Auch hier müssen Sie prüfen, ob der Rückgabewert von GetSaveFolderItem „Nil" ist, bevor Sie ihn verwenden (FolderItem wird Nil, wenn der Anwender auf „Abbrechen" klickt).

Im Beispiel wird eine „Sichern unter"-Dialogbox gezeigt, die als Namensvorschlag „Namenlos" enthält. Es liefert außerdem ein FolderItem, dessen Type und Creator auf den Dateityp „image/x-pict" passt, der für das Projekt in der Dateitypen-Dialogbox definiert wurde. Wenn der Anwender auf den Speichern-Knopf klickt, wird der Name, den der Anwender ausgewählt hat, angezeigt:

```
Dim f as FolderItem
f=GetSaveFolderItem("image/x-pict","Namenlos")
If f <> Nil Then
 MsgBox f.name
End if
```

Wenn Sie eine Textdatei erzeugen möchten, können Sie auch eine leere Zeichenkette statt des Dateityps als ersten Parameter übergeben. Die Methode CreateTextFile legt Dateityp und Creator sowieso automatisch an.

Wenn Sie die SaveAsDialog-Klasse verwenden, erzeugen Sie ein neues Objekt, das auf dieser Klasse basiert und weisen seinen Eigenschaften Werte zu, um das Erscheinungsbild festzulegen. Sie können folgende Eigenschaften steuern:

- Position (Eigenschaften Left und Top)
- Ausgangsverzeichnis (Eigenschaft Initial Directory)
- Dateinamen-Vorgabe (Eigenschaft SuggestedFileName)
- Gültige Dateitypen, die angezeigt werden sollen (Eigenschaft Filter)
- Beschriftung der Bestätigen- und Abbrechen-Knöpfe (Eigenschaften ActionButtonCaption & CancelButtonCaption)
- Text, der im Titelbalken des Dialogs erscheint (Eigenschaft Title)
- Text, der innerhalb des Dialogs erscheint (Eigenschaft PromptText)

Folgender Code öffnet eine benutzerdefinierbare „Sichern unter"-Dialogbox und zeigt den Inhalt des Dokumente-Verzeichnisses im Datei-Browser an.

```
Dim dlg as SaveAsDialog
Dim f as FolderItem
```

```
dlg=New SaveAsDialog
dlg.InitialDirectory=Volume(0).Child("Dokumente")
dlg.promptText="Geben Sie einen Dateinamen ein"
dlg.SuggestedFileName="RTF Datei"
dlg.Title="Speichern Sie Ihre Datei"
f=dlg.ShowModal()
If f <> Nil then
 //Datei gespeichert
Else
//Benutzerabbruch
End if
```

Mehr Informationen siehe „Welche Dateitypen gibt es?“ auf Seite 280 oder in der Sprachreferenz unter „GetSaveFolderItem“ und „SaveAsDialog in der Sprachreferenz“.

Benutzen von Textdateien

Textdateien sind Dateien, deren Dateityp TEXT ist. Sie können von Texteditoren (wie SimpleText oder BBEdit) oder von Textverarbeitungen (wie ClarisWorks) geöffnet werden. Textdateien sind sehr einfach in der Handhabung und man kann auch leicht überprüfen, was das eigene Programm in einer solchen Datei tatsächlich ablegt.

Für jeden lesenden oder schreibenden Zugriff auf eine Textdatei benötigen Sie ein FolderItem.

Lesen aus einer Textdatei

Nachdem Sie das FolderItem erzeugt haben, das auf die zu lesende Datei verweist, verwenden Sie die Methode „OpenAsTextFile“ der FolderItem-Klasse, um sie zu öffnen. Diese Methode öffnet einen Stream. Ein Stream ist ein Datenstrom, der Daten transportiert. In diesem Fall transportiert er die Textdaten aus Ihrer Datei in Ihr Programm. Deshalb wird er als TextInputStream bezeichnet. Diese Klasse ist darauf ausgerichtet, Textdateien zu lesen und stellt die Methoden ReadAll zum Lesen der kompletten Textdatei bis zur EOF-Marke (Ende der Datei) oder ReadLine zum zeilenweisen Auslesen, wobei jeweils das Carriage-Return-Zeichen ein Zeilenende markiert, zur Verfügung. Sie merkt sich auch die letzte Position, von der Sie aus der Datei gelesen haben.

Die Lesefunktionen liefern jeweils eine Zeichenkette. Das Ende der Datei wird in der Eigenschaft EOF des TextInputStream angezeigt. Hat diese den Wert True, wird es Zeit, die Datei mit Close zu schließen.

So lesen Sie eine Textdatei und zeigen ihren Inhalt in einem EditField an:

```
Dim f as FolderItem
Dim stream as TextInputStream
f=GetOpenFolderItem("text/plain")
If f<> Nil Then
 stream=f.OpenAsTextFile
 EditField1.text=stream.ReadAll()
 stream.Close
End if
```

Da ReadAll einfach den kompletten Inhalt der Datei einliest, könnte dies bei einer großen Datei zu Speichermangel führen, je nachdem, wieviel Speicher der Benutzer Ihrer Anwendung zugeordnet hat. Behalten Sie das immer im Hinterkopf, wenn Sie ReadAll einsetzen.

Das nächste Beispiel liest Textzeilen aus einer Datei im Preferences-Ordner in eine ListBox ein:

```
Dim f as FolderItem
Dim stream as TextInputStream
f = PreferencesFolder.child("My Apps Prefs")
If f<>Nil Then
  stream = f.OpenAsTextFile
  While Not stream.EOF
```

```
    ListBox1.addrow stream.ReadLine
  Wend
stream.Close
End If
```

Eine Textcodierung bestimmen

Sofern Sie die Textdateien nur mit REALbasic-Applikationen lesen und schreiben, wird das angeführte Beispiel funktionieren. Werden aber Dateien von anderen Applikationen bearbeitet oder stammen gar von anderen Plattformen oder aus anderen Ländern, werden Sie nicht umhin kommen, eine Textcodierung anzugeben. Andernfalls könnten die Zeichen falsch interpretiert werden. Wenn Sie Text einlesen, sollten Sie die Codierung des Textes angeben. Dies können Sie mit Hilfe der Encoding-Eigenschaft der TextEncoding-Klasse. Hier ist das abgeänderte Beispiel, bei dem eine Textcodierung angegeben wird.

```
Dim f as FolderItem
Dim stream as TextInputStream
f=GetOpenFolderItem("text/plain")
If f<> Nil Then
       stream=f.OpenAsTextFile
       Stream.Encoding=Encodings.MacRoman // MacRoman-Codierung angeben
       EditField1.text=stream.ReadAll()
       stream.Close
End if
```

Das Encodings-Objekt bietet Zugriff auf alle verfügbaren Codierungen. Verwenden Sie es immer dann, wenn Sie eine Codierung angeben müssen. Sie können die Codierung auch angeben, indem Sie diese als Parameter für die Read- und ReadAll-Methode angeben.

Weitere Informationen finden Sie im Abschnitt „Arbeiten mit Textcodierungen“ auf Seite 249.

Schreiben einer Textdatei

Mit der AppendToTextFile-Methode kann man in eine Datei schreiben. Existiert die Datei noch nicht oder soll eine existierende Datei überschrieben werden, verwendet man die CreateTextFile-Methode der FolderItem-Klasse. Diese Methoden sind Funktionen, die wieder einen Stream liefern, der diesmal ein TextOutputStream ist. Die TextOutputStream-Klasse enthält alle Funktionen, die man braucht, um in eine Textdatei zu schreiben. Der Text wird immer an das Ende einer existierenden Datei angehängt.

Die Methode WriteLine hängt standardmäßig ein Carriage-Return an das Ende jeder Zeile an. Mit der Eigenschaft Delimiter des TextOutputStreams können Sie ein beliebiges anderes Zeichen festlegen. Am Ende des Schreibvorgangs schließen Sie die Datei mit der Methode „Close“.

Hier zeigen wir, wie der Inhalt dreier EditFields in eine Textdatei geschrieben werden kann:

```
Dim file As FolderItem
Dim fileStream As TextOutputStream
file=GetSaveFolderItem("plain/text","Mein Info")
fileStream=file.CreateTextFile
fileStream.WriteLine namefeld.Text
fileStream.WriteLine addressefeld.Text
fileStream.WriteLine telefonfeld.Text
fileStream.Close
```

Festlegen einer Textcodierung

Wie beim Lesen kann es auch beim Schreiben vorkommen, dass Sie eine spezifische Textcodierung angeben müssen. Wenn die Applikation, die diese Datei liest, eine bestimmte Codierung erwartet, sollten Sie den Text vor dem Schreiben in diese Codierung konvertieren.

Dafür rufen Sie, bevor Sie den Text schreiben, die ConvertEncoding-Funktion auf. Hier ein Beispiel, das einen Text in MacRoman codiert.

```
Dim file As FolderItem
Dim fileStream As TextOutputStream
file=GetSaveFolderItem("plain/text","My Info")
fileStream=file.CreateTextFile
fileStream.Write ConvertEncoding(namefield.Text,Encodings.MacRoman)
fileStream.Close
```

Einschränkungen von Textdateien

Auf Textdateien kann nur in sequentieller Folge zugegriffen werden. Wenn man also etwas aus der Mitte der Datei lesen möchte, muss man erst alle Informationen davor lesen, bis man dort angekommen ist. Will man etwas in der Mitte der Datei einfügen, so muss man erst alles in die Datei schreiben, was davor kommt und dann den einzufügenden Teil und als Abschluss den Rest der Daten. Es ist ferner unmöglich, in eine Textdatei etwas zu schreiben, wenn aus dieser zugleich auch gelesen wird. Sollte so etwas in Ihrem Projekt vorkommen, müssen Sie zu binären Dateien greifen. Informationen dazu siehe „Binäre Streams“ auf Seite 297.

Styled Text

Unter Styled Text versteht man Textinformationen, die zusätzlich mit Angaben über Zeichensätze, Größen und Schriftarten versehen sind.

Styled Text in einem Eingabefeld

Mit einem FolderItem, das auf eine Datei mit Styled Text verweist, können Sie über die Methode OpenStyledEditField Daten aus dieser Datei lesen. Dieser Methode übergeben Sie den Namen eines EditFields, bei dem die Eigenschaft Styled den Wert True haben muss:

```
Dim f as FolderItem
f=GetOpenFolderItem("SimpleText-Dateien")
If f <> Nil Then
 f.OpenStyledEditfield EditField1
End if
```

Speichern von Styled Text

Wiederum über ein FolderItem schreiben Sie mit der SaveStyledEditField-Methode den Inhalt eines EditFields in eine Datei. Auch hier gilt wieder, dass die Eigenschaften Styled und MultiLine den Wert True haben müssen:

```
Dim f as FolderItem
f=GetSaveFolderItem("plain/text","Namenlos")
If f <> Nil Then
 f.SaveStyledEditField EditField1
End if
```

Arbeiten mit StyledText-Objekten

Eine andere Möglichkeit formatierten Text zu lesen und schreiben bietet StyledText-Klasse. Damit können Sie mit formatiertem Text arbeiten, der in keinem Zusammenhang mit einem EditField steht. Sie können die Formatierung bearbeiten und es auf Ihrer Festplatte speichern, um es später zu verwenden.

Sie arbeiten mit einem StyledText wie mit einer Folge von StyleRun-Objekten. Der StyleRun-Eintrag in der Sprachreferenz zeigt Ihnen, wie Sie mit Hilfe eines BinaryStream ein StyledText-Objekt speichern. Diesem Ansatz zufolge speichern Sie die StyleRun-Objekte in einem MemoryBlock und schreiben dann diesen Speicherblock mit Hilfe der Write-Methode der BinaryStream-Klasse auf die Festplatte. Für weitere Informationen lesen Sie bitte den Eintrag in der Sprachreferenz.

Verwenden von Bilddateien

Auch zum Öffnen und Speichern von Bitmaps und Vektorgrafiken hat REALbasic entsprechende Funktionen parat. Unter Mac OS unterstützt es Vektorgrafiken und gerasterte PICT-Dateien, unter Windows das .bmp-Format (Bitmap) und das .emf-Format (Extended Metafile Format). Bevor man eine PICT-Datei öffnen oder speichern kann, benötigt man ein FolderItem, das diese Datei repräsentiert. Die FolderItem-Methoden OpenAsPicture, SaveAsPicture und SaveAsJPEG stellen die benötigten Routinen bereit. Wenn Sie mit einer Vektorgrafik-Datei arbeiten, können Sie die OpenAsVectorPicture-Methode der FolderItem-Klasse verwenden. REALbasic versucht die Objekte der Datei in editierbare Object2D-Objekte umzuwandeln.

Speichern von Bildern

Um ein Bild in einer PICT-Datei abzuspeichern, benötigen Sie ein FolderItem, das eine neue oder eine bereits vorhandene PICT-Datei repräsentiert. Als nächstes rufen Sie die **SaveAsPicture**-Methode des FolderItems auf und übergeben dieser das Bild, das gespeichert werden soll. Im folgenden Beispiel wird das Backdrop-Bild eines Canvas-Steuerelements in einer Datei gespeichert, nachdem der Anwender in einer SaveAs-Dialogbox den Dateinamen festgelegt hat.

```
Dim f as FolderItem
f=GetSaveFolderItem("image/x-pict","Namenlos")
If f <> Nil Then
 f.SaveAsPicture Canvas1.backdrop
End If
```

Um das Bild im JPEG-Format zu speichern, ersetzen Sie den ersten Parameter für GetSaveFolderItem durch „image/jpeg“.

Die SaveAsPicture-Methode hat einen zweiten optionalen Parameter, über den Sie das Format der zu speichernden Datei festlegen können. Sie können sich entweder für eine Vektorgrafik- oder ein Bitmap-Format entscheiden. Sie können aber auch ein Meta-Format angeben. In der folgenden Tabelle sind die Codes für die Meta-Formate aufgelistet:

Tabelle 33. Codes und Beschreibung der Meta-Formate der SaveAsPicture-Methode

Wert	Klassenkonstante	Beschreibung
0	SaveAsMostCompatible	Das gebräuchlichste Format der jeweiligen Plattform (Mac: PICT, Win32: BMP)
1	SaveAsMostComplete	Format mit der höchsten Wahrscheinlichkeit für das Behalten aller Vektorinformationen (Mac: PICT, Win32: EMF)
2	SaveAsDefault	DefaultVector oder DefaultRaster, abhängig von den Bilddaten (Mac: PICT, Win32 Vector: EMF, Win32 Raster: BMP
3	SaveAsDefaultVector	Standard-Vektorformat der jeweiligen Plattform (Mac: PICT, Win32: EMF)
4	SaveAsDefaultRaster	Standard-Rasterformat der jeweiligen Plattform (Mac: Raster PICT, Win32: BMP)

Sie können im Code statt der numerischen Werte auch die Konstanten verwenden. Weitere Informationen zum Speichern im Vektor- oder Raster-Format finden Sie in der Sprachreferenz im Abschnitt über die FolderItem-Klasse.

Das Speichern eines Bildes, das in die Graphics-Eigenschaft eines Canvas-Steuerelements gezeichnet wurde, ist etwas komplizierter. Das kommt daher, dass die Eigenschaft Graphics kein PICT enthält. Daher muss man dem Fenster zusätzlich eine Picture-Eigenschaft verpassen. Alles, was nun in der Eigenschaft Graphics gezeichnet wird, muss ebenfalls in der Eigenschaft Picture gezeichnet werden. Dann kann das Bild auch über die SaveAsPicture-Methode gespeichert werden. Die Eigenschaft Picture, die Sie dem Fenster neu zugeordnet haben, muss zuerst mit einem leeren Bild belegt werden. Das erledigt die Funktion NewPicture am besten im Open-Event-Handler des Fensters. In unserem Beispiel ist die Eigenschaft Picture der Variablen p zugewiesen:

```
p=newpicture(canvas1.width,canvas1.height,32)
```

Im MouseDown-Event des Canvas-Steuerelements wird ein schwarzer Punkt gemalt, wenn der Benutzer mit der Maus klickt. Mit folgenden Zeilen übernimmt man diese Aktion auch in die Picture Eigenschaft p:

```
Me.Graphics.Pixel(x,y)=Rgb(0,0,0)
p.Graphics.Pixel(x,y)=Rgb(0,0,0)
```

Die Picture-Eigenschaft kann als PICT-Datei gesichert werden:

```
Dim f as FolderItem
f=GetSaveFolderItem("image/x-pict","Namenlos")
If f <> Nil Then
 f.SaveAsPicture p
End if
```

Öffnen von Bilddateien

Um eine Bilddatei zu öffnen, benötigen Sie das FolderItem der PICT- oder .bmp-Datei des Bildes. Dann können Sie das Bild mit der OpenAsPicture-Methode des FolderItems öffnen.

Folgender Beispielcode zeigt den „Datei öffnen“-Dialog, in dem der Anwender eine PICT- oder .bmp-Datei auswählen kann, die dann der Backdrop-Eigenschaft eines Canvas-Steuerelements zugewiesen wird.

```
Dim f as FolderItem
f=GetOpenFolderItem("image/x-pict")
If f <> Nil Then
 Canvas1.Backdrop=f.OpenAsPicture
End if

Dim f as FolderItem
f=GetOpenFolderItem("image/x-pict","image/x-bmp")
If f <> Nil Then
 Canvas1.Backdrop=f.OpenAsPicture
End if
```

Wenn es sich bei der Datei um eine Vektorgrafik handelt, können Sie zum Umwandeln der Dateiinformationen in für REALbasic verwertbare Vektorobjekte anstelle der OpenAsPicture-Methode die OpenAsVectorPicture-Methode aufrufen. Diese Option steht unter Mac OS für Vektor-PICT-Dateien und unter Windows für EMF-Dateien zur Verfügung.

Die Originaldatei enthält möglicherweise Objekte, für die es keine Entsprechung in REALbasic gibt. REALbasic versucht in diesem Fall, das bestmögliche Ergebnis bei der Umwandlung zu erzielen. In Abhängigkeit der Eigenschaften der Originaldatei kann es jedoch vorkommen, dass Informationen verloren gehen.

PICT-Dateien unterstützen Rechtecke, Linien, Ellipsen, RoundRects, Polygone, Text, Pixmaps und Bögen, die keiner Rotation ausgesetzt waren.

Extended Metafile-Dateien (.emf) unterstützen Rechtecke, Linien, Ellipsen, RoundRects, Polygone, Text, Pixmaps und Bögen, die keiner Rotation ausgesetzt waren.

.emf-Dateien werden in der Originalgröße wiedergegeben, die für die Bildschirmdarstellung meistens ungeeignet ist. Aller Wahrscheinlichkeit nach werden Sie vor dem Anzeigen der Datei die Größe herunterskalieren müssen (pic1.objects.scale=Skalierungsfaktor). Ein Skalierungsfaktor von 0.045 ist ein guter Wert.

Verwendung von Sounddateien

REALbasic kann Mac- und WAV-Sounddateien öffnen, aber nicht speichern. Mit der OpenAsSound-Methode des FolderItems einer Sounddatei kann diese wie folgt geöffnet und abgespielt werden:

```
Dim f as FolderItem
Dim s as Sound
f=GetFolderItem("Quack")
If f<> Nil Then
 s=f.OpenAsSound
 s.Play
End if
```

Töne, die in einer snd-Resource gespeichert sind, lassen sich ebenfalls verwenden. Mehr dazu siehe „Lesen von Resourcen“ auf Seite 301.

Verwenden von QuickTime-Movie-Dateien

Mit der OpenAsMovie-Methode des FolderItems eines QuickTime-Films lässt sich ein Film wie folgt abspielen:

```
Dim f as FolderItem
Dim m as Movie
f=GetOpenFolderItem("video/quicktime")
If f<> Nil Then
 m=f.OpenAsMovie
 moviePlayer1.Movie=m
 moviePlayer1.Play
End if
```

Wenn Ihr Programm einen bestimmten QuickTime-Film benötigt, können Sie ihn ins Projektfenster draggen und müssen nicht GetOpenFolderItem oder GetFolderItem verwenden.

Wenn Sie den Film innerhalb von REALbasic bearbeiten möchten, müssen Sie ihn als EditableMovie öffnen. Um den Film in einem MoviePlayer wiederzugeben, müssen Sie ihn der **Movie**-Eigenschaft des MoviePlayers zuweisen.

Das folgende Beispiel öffnet einen Film als EditableMovie und zeigt ihn anschließend in einem MoviePlayer-Steuerelement names **ThePlayer** an.

```
Dim f As FolderItem
Dim theEMovie as EditableMovie
f=GetOpenFolderItem("video/quicktime")
If f<>Nil and f.exists then
 theEMovie=f.OpenEditableMovie
 If theEMovie<>Nil then
  ThePlayer.movie=theEMovie
 end if
end if
```

Mit den Methoden der EditableMovie-Klasse können Sie:

- ein Segment in einem EditableMovie definieren,
- das Segment in die Zwischenablage ausschneiden oder kopieren,
- ein Segment in den aktuellen Film einfügen,

- dem aktuellen Film ein anderes Filmsegment anhängen,
- ein Segment an einer bestimmten Position im aktuellen EditableMovie einfügen,
- neue Sound- und/oder Video-Spuren anlegen,
- die Videospur skalieren.

Sie können die vorgenommenen Änderungen nach Beendigung automatisch oder durch Aufruf der CommitChanges-Methode speichern. Weitere Informationen zu Methoden, die Sie im Zusammenhang mit QuickTime-Filmen verwenden können, finden Sie in der Sprachreferenz im Abschnitt über EditableMovies.

Die Arbeit mit Binärdateien

In Binärdateien werden Daten im Binärformat und nicht als Text gesichert. Die Zahl 30000 würde als ASCII-Text fünf Zeichen (oder auch Bytes) benötigen, aber in einem Binärformat würde sie als Short Integer nur zwei Bytes belegen.

Im Unterschied zu Textdateien ist es möglich, auf eine Binärdatei gleichzeitig lesend und schreibend zuzugreifen. Auch der Direktzugriff auf eine bestimmte Position der Datei ist möglich, ohne dass alle davor stehenden Daten gelesen werden nüssen.

Die meisten Programme sichern ihre Daten im Binärformat. Um eine solche Datei lesen zu können, muss man wissen, wie die Daten in der Datei organisiert sind. Wurde die Datei von Ihrem eigenen Programm erzeugt, ist das kein Problem. Kommen die Daten von anderen Programmen, ist es schon schwieriger. Einige Formate werden veröffentlicht, wie beispielsweise das PICT-Format. Die meisten Softwarehersteller jedoch dokumentieren nicht öffentlich, wie die Dateien, die ihre Software erzeugt, aufgebaut sind.

Binäre Streams

Daten, die aus einer Binärdatei kommen, fließen über einen binären Stream. BinaryStream ist eine REALbasic-Objektklasse, die den Informationsfluss zwischen einem FolderItem und der von diesem FolderItem referenzierten Datei repräsentiert. Anders als bei TextInputStream und TextOutputStream, bei denen die Daten jeweils nur in eine Richtung befördert werden können, lässt ein BinaryStream sich zum Lesen und Schreiben von Daten verwenden. Daher muss man dem BinaryStream extra mitteilen, wenn man ihn nur zum Lesen verwenden will, so dass die Datei für Schreibzugriffe anderer Programme weiterhin zur Verfügung steht.

BinaryStreams können sowohl bestimmte Datentypen schreiben (wie String, Short Integer, Long Integer, Bytes). Sie können aber auch unformatierte Binärdaten schreiben.

Lesen einer Binärdatei

Mit einem FolderItem und der OpenAsBinaryFile-Methode bekommen Sie einen BinaryStream, aus dem Sie mit den Methoden Read, ReadByte, ReadLong, ReadPString und ReadShort Daten lesen können. Der BinaryStream merkt sich jeweils auch die Position innerhalb der Datei in der Eigenschaft Position. Durch Ändern dieser Eigenschaft können Sie an beliebige Stellen in der Datei springen.

Im folgenden Beispiel wird eine Datei geöffnet und eine Reihe von Zeichenketten (Strings) wird in eine ListBox eingelesen. Beachten Sie, dass der OpenAsBinary-Methode ein False übergeben wird, um die Datei im Read-Only-Modus zu öffnen, also nur lesend auf die Datei zuzugreifen. In einer Schleife werden die Daten gelesen, bis die Eigenschaft EOF den Wert True erreicht:

```
Dim f as FolderItem
Dim stream as BinaryStream
f=GetOpenFolderItem("MeinDateiTyp")
If f<> Nil Then
  ListBox1.DeleteAllRows
```

```
  stream=f.OpenAsBinaryFile(False)
  do
   ListBox1.AddRow stream.ReadPString
   ListBox1.Cell(ListBox1.ListCount-1,1)=stream.ReadPString
  Loop Until stream.EOF
 stream.Close
End if
```

Diese Schleife wäre etwa 25% schneller, wenn es sich um eine For…Next-Schleife handeln würde. Dazu müsste man aber schon vorher wissen, wieviele Zeilen gelesen werden müssen. Wäre in den ersten vier Bytes der Datei ein Long Integer-Wert untergebracht, der die Anzahl der Zeilen in der Datei angibt, dann könnte man so vorgehen:

```
Dim f as FolderItem
Dim stream as BinaryStream
Dim count,i as Integer
f=GetOpenFolderItem("MeinDateiTyp")
If f<> Nil Then
 ListBox1.DeleteAllRows
 stream=f.OpenAsBinaryFile(False)
 count=stream.ReadLong
 For i=1 to count
 Listbox1.AddRow stream.ReadPString
 Listbox1.Cell(ListBox1.Listcount-1,1)=stream.ReadPString
 Next
 stream.Close
End if
```

Unter Umständen müssen Sie auch beim Lesen eines BinaryStreams die Codierung der Zeichen berücksichtigen. Dazu können Sie der Read- und ReadPString-Methode die Codierung als optionalen Parameter übergeben. Verwenden Sie dazu das Encodings-Objekt. Weitere Informationen finden Sie im Abschnitt „Arbeiten mit Textcodierungen“ auf Seite 249.

Schreiben einer Binärdatei

Mit einem FolderItem und der Methode „OpenAsBinaryFile“ kann eine existierende Datei geöffnet werden. Mit CreateBinaryFile wird eine neue Datei angelegt. Beide Methoden liefern einen BinaryStream zurück, über welchen mit den Methoden Write, WriteByte, WriteLong, WritePString und WriteShort Daten in die Datei geschrieben werden können. Der BinaryStream merkt sich in der Eigenschaft Position auch die Position innerhalb der Datei. Durch Ändern dieser Eigenschaft können Sie an eine beliebige Position in der Datei springen.

Sind die Schreibzugriffe abgeschlossen, schließen Sie die Datei mit der Close-Methode des BinaryStream.

Im Beispiel wird der Inhalt zweier Spalten einer Listbox in eine Datei geschrieben und dann wird die Datei geschlossen. Dieses Beispiel erzeugt die Datei, die im obigen Beispiel mit der For…Next-Schleife eingelesen wurde:

```
Dim f as FolderItem
Dim i as Integer
Dim stream as BinaryStream
f=GetSaveFolderItem("MeinDateiTyp","Namenlos")
If f<> Nil Then
 stream=f.CreateBinaryFile("myFileType")
 stream.WriteLong ListBox1.ListCount
 For i=0 to ListBox1.Listcount-1
 stream.WritePString ListBox1.List(i)
 stream.WritePString ListBox1.Cell(i,1)
 Next
 stream.Close
End if
```

Virtuelle Volumes

Bis zu diesem Punkt drehte es sich in diesem Kapitel um herkömmliche Schreibtisch-Dateien und -Ordner, die von jeder Anwendung, die für einen bestimmten Dateityp ausgelegt ist, erstellt, ausgelesen oder verändert werden können. REALbasic ermöglicht es Ihnen zusätzlich, eine besondere Art von Dokument, das mehrere verschiedene Dateitypen vereinen kann, anzulegen und zu pflegen. Unter diesem Gesichtspunkt handelt es sich dabei eher um ein Volume als um eine einzelne Datei. Aus diesem Grunde spricht man auch von „virtuellen Volumes". Ein virtuelles Volume kann von einer REALbasic-Anwendung angelegt werden. Die Anwendung kann auf das Volume schreiben und es auslesen. Virtuelle Volumes werden von allen Plattformen unterstützt. So können Sie beispielsweise auf einem Mac ein virtuelles Volume anlegen und es dann unter Windows auslesen, bestehende Dateien verändern oder zusätzliche Dateien erzeugen.

Über die VirtualVolume-Klasse können Sie eine Hierarchie von „virtuellen" Dateien innerhalb einer tatsächlich vorhandenen Datei anlegen und pflegen. Die VirtualVolume-Klasse unterstützt das Lesen und Schreiben von Text- und Binary-Streams. Das bedeutet, dass Sie über die TextOutputStream- und die BinaryStream-Klassen Text- und Binär-Dateien erzeugen können. Das bedeutet aber auch, dass die SaveAsPicture- und SaveStyledEditField-Methoden der FolderItem-Klasse mit virtuellen Volumes nicht funktionieren.

Die FolderItem-Klasse besitzt zwei Methoden und eine Eigenschaft für das Arbeiten mit virtuellen Volumes. Verwenden Sie die CreateAsVirtualVolume- und die OpenAsVirtualVolume-Methoden, um ein virtuelles Volume anzulegen und es zu öffnen. Falls sich ein bestimmtes FolderItem auf einem virtuellem Volume befindet, enthält seine VirtualVolume-Eigenschaft das virtuelle Volume.

Wenn Sie einmal ein virtuelles Volume angelegt haben, können Sie die Root-Eigenschaft auslesen und das Wurzelverzeichnis (root) mit den gleichen FolderItem-Methoden, die Sie auch für „echte" Dateien einsetzen, ansteuern. Virtuelle Volumes unterstützen keine Resource Forks.

Der folgende Code legt ein virtuelles Volume an und ruft die Root-Eigenschaft auf. Wenn Sie einmal das Wurzelverzeichnis (root) des virtuellen Volumes kennen, können Sie mit Hilfe der TextOutputStream- und BinaryStream-Klassen Dateien relativ zum Wurzelverzeichnis anlegen.

```
Dim f as FolderItem
Dim v as VirtualVolume
v=New VirtualVolume
f=GetFolderItem(“”).Child(“VV“)
If f<>Nil then
v=f.CreateVirtualVolume
if v.Root<> Nil then
MsgBox v.Root.Name
End if
End if
```

Verwenden von Macintosh Resources

Alle Macintosh-Dateien (inklusive der Programme, die ja im Grunde auch nur Dateien sind) bestehen aus zwei Teilen, die Forks genannt werden. Der Data-Fork enthält die Daten im applikationsspezifischen Format und der Resource-Fork enthält formatierte Informationen zu der Datei, wie Icons, Töne, Menüleisten, Bilder, Listen von Strings etc.

REALbasic erlaubt das Lesen und Schreiben des Resource-Fork. Auch dies wird über ein FolderItem realisiert. Mehr zu diesem Thema bietet zum Beispiel das Buch *Inside Macintosh: Resources* von Addison-Wesley (englisch).

REALbasic unterstützt das Arbeiten mit dem Resource-Fork nur auf der Macintosh-Plattform. Die Beschreibung und die Bespiele in diesem Abschnitt setzen voraus, dass das Programm auf einem Macintosh läuft. Wenn Ihr Programm Macintosh-Resources verwenden soll, können Sie die Konstanten TargetWin32 oder TargetMacOS verwenden, um sicherzustellen, dass das Programm auf einem Macintosh läuft, bevor Sie auf den Resource-Fork zugreifen. Ausnahme: Es ist

möglich, benutzerdefinierte Mauszeiger in Ihr Programm einzubinden, indem Sie die CURS-Resource verwenden und auf diese in der erzeugten Windows-Applikation zugreifen. Dies wird im Abschnitt „Benutzerdefinierte Maus-Cursor in Windows-Applikationen“ auf Seite 301 beschrieben.

Öffnen des Resource-Forks

Mit dem FolderItem und der Methode OpenResourceFork erhält man ein Objekt der Klasse ResourceFork oder „Nil“, wenn die Datei keinen Resource-Fork enthält (z.B. weil sie von einem PC kommt).

Das Beispiel ermittelt, wie viele Resource-Objekte in einem Resource-Fork enthalten sind:

```
Dim f as FolderItem
Dim rf as ResourceFork
f=GetOpenFolderItem("any")
If f <> Nil Then
 rf=f.OpenResourceFork
 If rf=Nil Then
 Beep
 MsgBox "Die Datei hat keinen Resource-Fork."
 Else
 MsgBox "Die Datei hat"+str(rf.TypeCount)+" Resource-Tpyen."
 End if
End if
```

Der Dateityp „any“, der an die GetOpenFolderItem-Dialogbox übergeben wird, wurde im Dateitypen-Dialog zuvor definiert und verwendet den String „????“ für Type und Creator. Die Fragezeichen passen auf jeden Type- oder Creator-Code.

Anlegen eines Resource-Fork

Bevor man in einen Resource-Fork schreiben kann, muss die Datei zunächst einen haben. Über OpenResourceFork stellt man daher zunächst fest, ob die Datei einen Resource-Fork hat. Ist das nicht der Fall, kann er mit CreateResourceFork angelegt werden. Das Beispiel zeigt, wie das geht:

```
Dim f as FolderItem
Dim rf as ResourceFork
f=GetOpenFolderItem("any")
If f <> Nil Then
 rf=f.OpenResourceFork
 If rf=Nil Then
 rf=f.CreateResourceFork("any")
 End if
End if
```

Einem Projekt einen Resource-Fork hinzufügen

Sie können einem REALbasic-Projekt einen Resource-Fork hinzufügen, indem Sie eine Resource-Datei in das Projektfenster draggen. REALbasic erkennt eine Resource-Datei am Datei-Typ „rsrc“. Sie können beliebig viele Resource-Dateien in Ihr Projekt einbinden. Die Resourcen aller Resource-Dateien werden in die erzeugte Applikation eingebunden. Im Falle eines Konflikts überschreiben spätere Resource-Dateien frühere, und zwar in der Reihenfolge, in der sie im Projektfenster erscheinen. Folgendes Beispiel öffnet den Resource-Fork der Applikation:

```
Dim rf as ResourceFork
rf=App.ResourceFork
```

Unterstützte Resource-Typen

REALbasic kennt Funktionen für PICT-, CICN-, CURS-, und snd-Resourcen. Mit der Methode AddPicture kann eine PICT-Resource eingefügt werden und mit GetPicture oder GetNamedPicture kann man PICT-Resourcen aus einem Resource-Fork lesen. Mit GetCicn greift man auf die ColorIcon-Resource zu. Töne kann man aus der snd-Resource über GetSound erhalten. Jede andere Form von Resource-Daten kann in Form von binären Daten aus dem Resource-Fork geholt bzw. in ihn geschrieben werden. Sie müssen dann aber wissen, wie diese Daten aufgebaut sind, damit ihr Programm sie interpretieren kann.

Lesen von Resourcen

Die ResourceFork-Klasse besitzt Methoden, um Daten aus Resourcen vier verschiedener Typen zu lesen. Sie können PICT-Resourcen mit den Methoden GetPicture und GetNamedPicture lesen. Sie können ein Farb-Icon-Picture durch Aufruf der Methode GetCicn erhalten. Sie können ein großes (32*32) Icon über die GetIcl-Methode und ein kleines (16*16) Icon über die GetIcs-Methode bekommen. Das Beispiel zeigt eine Picture-Resource in einem ImageWell an:

```
Me.Image=App.ResourceFork.GetPicture(128)
```

Auf die gleiche Art können Sie GetCicn, GetIcl und GetIcs verwenden.

Sie können Sounds aus „snd "-Resourcen mit der GetSound-Methode der ResourceFork-Klasse laden.

Lesen benutzerdefinierter Maus-Cursor

Verwenden Sie GetCursor, um der MouseCursor-Eigenschaft der Applikation, einem Fenster oder einem Steuerelement einen benutzerdefinierten Maus-Cursor zuzuweisen. Als Beispiel weist folgender Code im MouseEnter-Event-Handler eines Fensters der MouseCursor-Eigenschaft des Fensters einen benutzerdefinierten Maus-Cursor zu:

```
self.MouseCursor=App.ResourceFork.GetCursor(128)
```

Diese Zeile bewirkt, dass REALbasic den benutzerdefinierten Maus-Cursor immer dann anzeigt, wenn der Mauszeiger den Bereich des Fensters berührt. Soll sich der Maus-Cursor ändern, wenn sich die Maus über einem Steuerelement in diesem Fenster befindet, können Sie folgende Zeile im MouseEnter-Event-Handler des Steuerelements verwenden:

```
self.MouseCursor=App.ResourceFork.GetCursor(129)
```

und dann den benutzerdefinierten Maus-Cursor des Fensters mit folgender Zeile im MouseExit-Event-Handler des Steuerelements wiederherstellen:

```
self.MouseCursor=App.ResourceFork.GetCursor(128)
```

Beachten Sie, dass diese Zeilen der MouseCursor-Eigenschaft des Steuerelements keinen Cursor zuweisen, sie ändern nur die Zuweisung zum Elternfenster des Steuerelements.

Hinweis: Wenn das Elternfenster des Steuerelements oder die Applikation selbst eine benutzerdefinierte MouseCursor-Eigenschaft besitzt, wird die MouseCursor-Eigenschaft des Steuerelements ignoriert.

Der ganzen Applikation können Sie einen benutzerdefinierten Cursor zuweisen, indem Sie folgenden Code in den Open-Event-Handler des App-Objekts eintragen:

```
MouseCursor=App.ResourceFork.GetCursor(128)
```

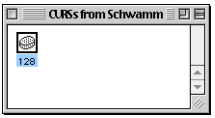
Dies führt dazu, dass die REALbasic-Applikation die ganze Zeit diesen benutzerdefinierten Cursor verwendet und die MouseCursor-Eigenschaften aller Fenster und Steuerelemente der Applikation ignoriert werden.

Benutzerdefinierte Maus-Cursor in Windows-Applikationen

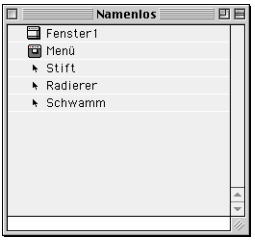
Sie können CURS-Resourcen dazu verwenden, benutzerdefinierte Maus-Cursor in erzeugten Windows-Applikationen zuzuweisen. Dies ist derzeit der einzige Fall, in dem Resourcen in Programmen, die für Windows erzeugt wurden, unterstützt werden. Die Beziehungen zwischen den MouseCursor-Eigenschaften der Application, der Fenster und der Steuer-

elemente sind die gleichen wie bei Macintosh-Applikationen Sie müssen die Resource-Datei jedoch auf eine bestimmte Weise erzeugen. Diese Technik empfehlen wir auch für Macintosh-Applikationen.

Für jeden benutzerdefinierten Cursor, den Sie im Projekt verwenden möchten, müssen Sie eine eigene Resource-Datei erzeugen. Jede Resource-Datei darf nur eine CURS-Resource beinhalten. Normalerweise weisen Sie den benutzerdefinierten Cursor der ID 128 zu. Hier ein Beispiel:



Danach draggen Sie alle diese Resource-Dateien in Ihr Projekt. Diese speziellen Resource-Dateien erscheinen im Projektfenster mit einem Cursor-Icon wie in folgender Abbildung:



Sie können dann auf die benutzerdefinierten Cursor-Ressourcen über ihre Namen zugreifen. Als Beispiel ändert folgender Befehl im MouseEnter-Event-Handler eines Steuerelements den Mauszeiger in den Schwamm-Cursor, wenn die Maus den Bereich des Steuerelements betritt:

```
Self.MouseCursor=Sponge
```

Sie sollten dann den alten Cursor mit dem passenden Befehl im MouseExit-Event-Handler des Steuerelements wiederherstellen.

Lesen von anderen Resourcen

Um Daten aus anderen Resourcen zu lesen, müssen Sie das Format der Resource kennen. Wenn Sie z.B. die STR#-Resource lesen möchten, können Sie dies mittels der GetResource-Methode der ResourceFork-Klasse tun. Sie erhalten dann die Anzahl der Bytes, die die Resource mit der genannten ID (in diesem Fall STR#) enthält. Wenn Sie dann berücksichtigen, dass die ersten zwei Bytes der STR#-Resource die Anzahl der Strings in der Resource repräsentieren und die Strings im Pascal-Format vorliegen (erstes Byte ist die Länge des Strings), dann können Sie schon etwas sinnvolles damit anstellen.

Wenn Sie Ihre Applikation erzeugen, können Sie Versionsinformationen über die Applikation in den „Applikation erzeugen“-Dialog eingeben. Diese Informationen werden in einer „vers“-Resource gespeichert, die Teil Ihrer Applikation wird. Eine Applikation kann auf die „vers“-Resource mit der GetResource-Methode der ResourceFork-Klasse zugreifen. Welche Informationen in die „vers“-Resource geschrieben werden, erfahren Sie im Abschnitt „Versions-Information“ auf Seite 390.

Schreiben von Resourcen

Mit der Methode AddPicture legt man ein neues Bild in eine vorhandene PICT-Resource. Mit der AddResource-Methode legt man neue Resourcen an und mit RemoveResource kann eine bestimmte Resource gelöscht werden. Um Veränderungen an Nicht-PICT-Resourcen vorzunehmen, holen Sie sich zunächst die Daten über GetResource, löschen diese dann mit RemoveResource und hängen die Daten erneut mittels AddResource an. Mehr Information darüber siehe ResourceFork in der Sprachreferenz.

Über den Schreibtisch geöffnete Dateien

Führt der Benutzer einen Doppelklick auf eine Datei aus, deren Creator-Code mit dem Creator-Code Ihres Programms übereinstimmt, oder zieht er die Datei auf das Icon Ihres Programms, wird er zurecht erwarten, dass dann Ihr Programm gestartet und die Datei geöffnet wird. Um dies zu realisieren, benötigt man eine neue Klasse, die auf der Application-Klasse beruht. Diese Klasse steht für Ihre Anwendung und empfängt Informationen, wenn der Benutzer einen Doppelklick auf eine Datei mit Ihrer Datei-Kennung ausführt. Die Application-Klasse stellt dafür den Event-Handler OpenDocument zur Verfügung, der in diesem Fall ausgeführt wird. Er bekommt als Parameter ein FolderItem, das die Datei, auf die doppelt geklickt wurde, repräsentiert. Gehen Sie so vor:

1. Wenn Ihr Projekt noch keine Application-Klasse verwendet, wählen Sie **Datei/Neue Klasse**.

   Hinweis: Jedes Projekt, das über die Projektvorlage angelegt wird, enthält bereits eine auf der Application-Klasse beruhende App-Klasse.
2. Im Eigenschaftenfenster wählen Sie im Popup **Super** den Typ **Application**.
3. Als Name geben Sie dort **App** ein.
4. Klappen Sie im Code-Editor die Event-Liste auf.
5. Klicken Sie auf das Event **OpenDocument**.
6. Geben Sie hier den Code zum Öffnen des doppelt angeklickten Dokuments ein.

REALbasic behandelt eine Datei, die auf das Icon Ihrer Anwendung „fallengelassen“ wird, genauso wie eine Datei, auf die der Benutzer doppelt geklickt hat.

Anlegen neuer Dateien

Startet ein Benutzer Ihr Programm, ohne es über eine Datei zu öffnen, kann es sein, dass Sie gern automatisch ein neues Dokument bereitstellen würden. Dazu können Sie den Event-Handler „NewDocument“ der App-Klasse verwenden. Unter Macintosh wird dieser auch ausgeführt, wenn die Anwendung ein OpenApplication-AppleEvent (oapp) erhält, das ausgelöst wird, wenn AppleScript dem Finder sagt, dass Ihr Programm gestartet werden soll.

Den NewDocument-Event-Handler können Sie mit „NewDocument“ aufrufen, wodurch Sie nur an einer Stelle die nötigen Schritte ablegen müssen, die für das Erzeugen eines neuen Dokuments abgearbeitet werden müssen.

Kapitel 9 Wiederverwendbare Objekte durch Klassen

Klassen dienen als Vorlagen für Objekte, so wie beispielsweise die im Projektfenster aufgeführten Fenster als Vorlagen für die Fenster in Ihrem Programm verwendet werden. Dieses Kapitel wird Ihnen die Vorzüge von Klassen näherbringen und zeigen, wie man sie verändert und wie man selbstdefinierte Bedienelemente mit Klassen anlegt.

Inhalt

- Die Vorzüge von Klassen
- Definition von Unterklassen
- Zugriff innerhalb der Klasse auf die Eigenschaften und Methoden dieser Klasse
- Verändern von Klassen
- Behandlung von Menüs in Klassen
- Verwendung von Klassen im eigenen Projekt
- Die Application-Klasse
- Anlegen selbstdefinierter Steuerelemente mit Klassen
- Virtuelle Methoden
- Klasseninterface
- Interface-Vererbung
- Selbstdefinierte Objektverknüpfungen

Die Vorzüge von Klassen

Wiederverwendbarer Code

Wird das Verhalten eines PushButtons programmiert, dann kann der dazu geschriebene Programmteil nur für diesen PushButton verwendet werden. Will man dieses Programmstück auch für einen anderen PushButton verwenden, dann muss man es kopieren und so anpassen, dass es den Namen des neuen PushButtons enthält, sofern es sich vorher auf den Original-PushButton bezogen hat.

Für Klassen hingegen wird dieses Programmstück nur einmal definiert. Die Referenz auf das eigentliche Objekt (hier eben ein PushButton) erfolgt symbolisch. Das gleiche Programmstück kann so ohne Änderung immer wieder verwendet werden. Wenn Sie eine Klasse auf Basis eines PushButton-Steuerelements erzeugen und dann dieser Klasse Ihren Code hinzufügen, besitzt jede Instanz dieser selbstdefinierten Klasse diesen Code.

Kleinere Projekte und Programme

Weil die Klassen es erlauben, Programmteile einmal abzulegen und dann innerhalb des Projektes immer wieder zu verwenden, wird das Gesamtprogramm kürzer und braucht später wahrscheinlich auch weniger Speicher.

Erleichterte Programmwartung

Kürzere Programme bedeuten auch immer weniger Wartungsaufwand. Wenn Sie ein ähnliches Programmstück an mehreren Stellen im Programm abgelegt haben, dann müssen Sie das immer im Hinterkopf behalten, wenn Sie sich an die Behebung von Fehlern machen oder das Projekt verändern. Ist der Programmteil aber nur einmal vorhanden, werden Sie logischerweise auch erheblich weniger Zeit mit Änderungen verbringen, weil die Suche nach Stellen im Programm, an denen ähnliche Teile verwendet wurden, entfällt. Die Änderung der Klasse wirkt sich sofort auf alle Stellen aus, an denen sie verwendet wurde.

Erleichterte Fehlersuche

Binsenweisheit: Je weniger Programmtext existiert, desto weniger Code muss nach Fehlern durchforstet werden.

Mehr Einfluss

Klassen eröffnen Ihnen mehr Einflussmöglichkeiten, als Sie durch das Einfügen von Code in den Event-Handler eines Steuerelements haben. Tatsächlich können Klassen sogar die Steuerung von Menüs übernehmen. Ferner können Sie Klassen anlegen, mit denen Sie eigene Steuerelemente definieren. Klassen haben darüber hinaus den Vorzug, dass Sie eine Version der Klasse anlegen können, die den Zugriff auf den Quelltext unterbindet. Dadurch können Sie Klassen mit anderen REALbasic-Programmierern austauschen oder auch verkaufen.

Insgesamt wird das Programmieren durch Verwenden von Klassen erheblich effizienter.

Definition von Instanzen

REALbasic besitzt viele eingebaute Klassen. PushButton, StaticText, EditField oder ListBox sind, um nur einige Beispiele zu nennen, Klassen von Steuerelementen. Solche Steuerelement-Klassen kann man als Vorlagen für die Objekte verstehen, die Sie in dem GUI Ihrer Applikation verwenden. Sie sind eher abstrakt zu sehen, da sie selbst nie verwendet werden. Stattdessen werden Instanzen dieser Klassen erzeugt und verwendet.

Wenn Sie beispielsweise ein EditField in ein Fenster ziehen, erzeugen Sie eine Instanz der EditField-Klasse. Diese Instanz enthält alle Eigenschaften und Methoden der Vorlage. Dieses Prinzip gilt für alle weiteren Klassen. Sie passen die Instanz Ihren Bedürfnissen an, indem Sie deren Eigenschaften modifizieren.

Definition von Unterklassen

Manchmal werden Sie eine Klasse in einer Variante benötigen, die gegenüber der Definition leicht modifiziert ist. Beispielsweise könnte es nötig sein, dass Sie ein EditField benötigen, bei dem Ausschneiden (Cut) und Kopieren (Copy) im Menü deaktiviert sind, damit der Benutzer bestimmte Daten nicht ins Klemmbrett befördern kann (z.B. ein Passwort). Oder Sie brauchen eine ListBox, die als Grundeinstellung die Monate eines Jahres anzeigt. Solche Modifikationen der eingebauten Klassen können Sie mithilfe einer Unterklasse definieren, die Sie Ihrem Projekt hinzufügen.

Es gibt einen gravierenden Unterschied, ob Sie eine Klasse instanziieren, zum Beispiel, indem Sie ein Element in ein Fenster ziehen, oder eine Unterklasse ableiten und Ihrem Projekt hinzufügen. Eine abgeleitete Klasse können Sie wiederum instanziieren und einem Fenster hinzufügen. Sie können die abgeleitete Klasse auch speichern und in anderen Projekten verwenden.

Was ist eine Unterklasse?

Eine Unterklasse ist eine Klasse mit einer übergeordneten Oberklasse. Die Oberklasse ist die Basis der Unterklasse und wird auch „parent“ (Elternteil) genannt. Die Unterklasse wird von der Oberklasse abgeleitet. Unterklassen beherbergen

alle Eigenschaften, Methoden und Events ihrer Oberklasse, können sie aber verändern. Zunächst sind die Unterklassen mit den Oberklassen jedoch identisch, bis Sie anfangen, sie zu modifizieren. Danach bestehen die einzigen Unterschiede zur Oberklasse eben genau in den Punkten, in denen Sie Methoden, Eigenschaften oder Events verändert oder hinzugefügt haben.

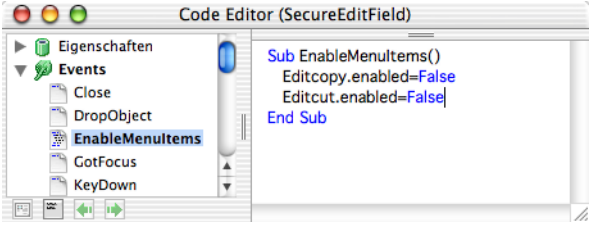
Beispiele von Unterklassen

Nehmen wir an, dass Sie ein EditField definieren, bei dem Ausschneiden (Cut) und Kopieren (Copy) im Menü deaktiviert werden (nennen wir es mal SecureEditField – sicheres Eingabefeld). Dazu legen Sie eine neue Klasse an und wählen das EditField als seine Oberklasse (super class). Die neue Unterklasse ist in folgender Abbildung dargestellt:

Abb. 224: Sicheres Eingabefeld basierend auf der EditField-Klasse

REALbasic aktiviert automatisch die Menüpunkte **Ausschneiden** und **Kopieren**, sobald Text im Eingabefeld selektiert wird. Weil der Fokus in EditFields gesetzt werden kann, hat jede Unterklasse des EditField-Objekts automatisch einen EnableMenuItems-Event-Handler. Dies gestattet dem EditField, Einfluss auf die Menüs auszuüben, wenn der Fokus sich im EditField befindet. Um nun die Aktivierung der Menüpunkte Ausschneiden und Kopieren zu verhindern, müssen Sie im Event-Handler **EnableMenuItems** unseres neuen SecureEditFields die Eigenschaft **Enabled** für diese Menüpunkte auf **False** setzen.

Abb. 225: Kopieren- und Ausschneiden-Menüpunkte im SecureEditField deaktivieren



Um eine Instanz des SecureEditFields einzusetzen, draggen Sie es einfach aus dem Projektfenster in einen Fenstereditor. Auch wenn es aus dem Projektfenster und nicht aus der Steuerelementepalette auf den Fenstereditor gezogen wurde, sieht es wie jedes andere EditField aus. Es verhält sich wie ein normales EditField, außer dass die Menüpunkte **Kopieren** und **Ausschneiden** deaktiviert sind, wenn der Anwender Text selektiert hat.

Abb. 226: Eine Instanz eines SecureEditFields kann selektierten Text nicht kopieren.

Da das SecureEditField im Projektfenster aufgeführt wird, können Sie davon an der jeder beliebigen Stelle Instanzen erzeugen und seinen Code an einer zentralen Stelle belassen und bearbeiten.

Angenommen, Sie brauchen eine ListBox, die als Grundeinstellung alle Monate anzeigt und bei der der aktuelle Monat die voreingestellte Auswahl ist. Legen Sie zunächst eine neue Klasse an, die ListBox als Oberklasse hat. Im Open-Event-Handler Ihrer neuen Unterklasse verwenden Sie die Methode AddRow, um die Namen der Monate in die Box einzufügen. Danach schreiben Sie die benötigten Programmzeilen, um den aktuellen Monat zu selektieren.

Soll ein EditField angelegt werden, in das der Benutzer nur Zahlen eingeben kann, geht man so vor: Legen Sie eine Unterklasse der EditField-Oberklasse an und nennen Sie diese „NumbersOnlyEditField“. Im Event „KeyDown“ legen Sie den Programmteil ab, der Zahlen annimmt, aber alle anderen Zeichen ignoriert. Ist diese Unterklasse einmal definiert, können Sie auch an anderen Stellen des Projekts verwenden. Der Programmtext existiert aber nur genau einmal.

Unterklassen sind normale Klassen. Sie heißen nur Unterklassen, um sie von den vordefinierten Klassen zu unterscheiden und um zu betonen, dass sie die Eigenschaften, Events und Methoden der Eltern erben. Folgerichtig kann eine Unterklasse wiederum als Oberklasse für eine neue Unterklasse dienen. Wird ein Feld benötigt, das nur Zahlen eines bestimmten Wertebereiches annimmt, legt man eine Unterklasse der gerade definierten Klasse „NumbersOnlyEditField“ an. Die neue Unterklasse hat zunächst wieder die identische Definition von „NumbersOnlyEditField“. Im Event-Handler „TextChanged“ definiert man nun aber zusätzlich noch, welcher Wertebereich akzeptiert werden soll.

Man hätte genauso ein „NumbersOnlyEditField“ kopieren können und den dort abgelegten Programmtext verändern können. Allerdings hätte man dadurch zusätzlichen Programmtext erzeugt und hätte sich später bei Änderungen am „NumbersOnlyEditField“ daran erinnern müssen, dass es eine ähnliche Kopie des Programmtextes noch bei einem anderen Objekt gibt, bei dem man die gleichen Änderungen nochmals durchführen muss.

Zugriff auf Eigenschaften und Methoden innerhalb einer Klasse

Indem Sie ein Steuerelement, wie z.B. einen PushButton, um eigenen Code erweitern, fügen Sie diesen Code genau genommen zu einer Instanz der PushButton-Klasse hinzu. Deshalb muss in Ihrem Code eine Referenz auf genau diese Instanz enthalten sein, da REALbasic sonst nicht feststellen könnte, auf welchen PushButton sich Ihr Code bezieht.

Beim Hinzufügen von Code zu einer Klasse oder Unterklasse ist es nicht notwendig, einen Bezug zu einer bestimmten Instanz herzustellen, denn der Code wird Teil der Klasse, in die er eingefügt wird und nicht Teil der Instanz. Also referiert man innerhalb des Codes einer Klasse nicht auf Objekte dieser Klasse. Angenommen, Sie legen in einem Fenster einen PushButton namens PushButton1 an, der deaktiviert werden soll, nachdem der Benutzer ihn angeklickt hat. Der Code im Event-Handler des PushButtons würde dann so aussehen:

```
PushButton1.Enabled=False
```

Hätte man statt dessen eine Unterklasse angelegt, deren Oberklasse PushButton ist, würde man nicht auf die Instanz referieren und der Code wäre daher:

```
Enabled=False
```

Wird eine Unterklasse benutzt, bezieht sich der Code automatisch auf die Instanz, die gerade benutzt wird.

Klassen anlegen

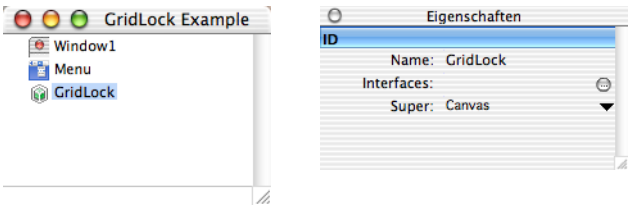
Möchten Sie eine Klasse von einer bereits existierenden Klasse ableiten, so befolgen Sie die folgenden Schritte.

1. Mit dem Projektfenster im Vordergrund wählen Sie **Ablage/Neue Klasse.**

   Dem Projektfenster wird eine neue Klasse mit dem Namen **Klasse1** hinzugefügt. Wird die neue Klasse ausgewählt, zeigt das Eigenschaftenfenster die Eigenschaften der neuen Klasse an. Standardmäßig wird die neue Klasse nicht von einer anderen Klasse abgeleitet.
2. Soll die neue Klasse die Eigenschaften einer bestehenden Klasse übernehmen, müssen Sie dazu im Eigenschaftenfenster die Super-Eigenschaft auf die Klasse setzen, von der die neue Klasse die Eigenschaften übernehmen soll.
3. Verwenden Sie das Eigenschaftenfenster, um der neuen Klasse einen Namen zu geben.

   Das „Gridlock"-Beispiel auf der CD verwendet ein herkömmliches Steuerelement, um ein rechteckiges Gitter in ein Fenster zu zeichnen. In Abbildung 227 sehen Sie, dass dieses Steuerelement auf einem Canvas-Steuerelement basiert.

Abb. 227: Die Gridlock-Klasse basiert auf dem Canvas-Steuerelement



Wenn Sie die neue Klasse im Projektfenster ausgewählt haben, können Sie sie doppelklicken, um den Code-Editor für die neue Klasse zu öffnen.

Unterklassen aus vorhandenen Klassen ableiten

Wenn Sie eine Unterklasse aus einer bereits im Projekt vorhandenen Klasse erzeugen wollen, gibt es dafür zwei Wege:

Um eine Unterklasse einer bestehenden Klasse zu erzeugen, unternehmen Sie folgendes:

1. Halten Sie die ctrl-Taste (Windows: rechte Maustaste) gedrückt und klicken Sie auf die Klasse, von der Sie eine Unterklasse erzeugen möchten.
2. Wählen Sie **Neue Unterklasse** aus dem Kontextmenü. Dem Projekt wird eine neue Klasse hinzugefügt. Im Eigenschaftenfenster wird die bestehende Klasse als Eltern (Super)-Klasse der neuen Klasse eingesetzt.
3. Verwenden Sie das Eigenschaftenfenster, um der Klasse einen neuen Namen zu geben.

Um eine Unterklasse einer Steuerelement-Klasse zu erzeugen, unternehmen Sie folgendes:

1. Draggen Sie das Steuerelement in das Projektfenster. Dadurch wird dem Projekt eine Klasse hinzugefügt, die auf dem Steuerelement basiert.
2. Verwenden Sie das Eigenschaftenfenster, um der Klasse einen neuen Namen zu geben.

Klassen sichern

Da Klassen wiederverwendbar sind, werden Sie eine Bibliothek von Klassen anlegen wollen, die Sie ohne Mühe in andere Projekte importieren können. Um eine Klasse zu sichern, müssen Sie diese nur aus dem Projektfenster auf den Schreibtisch ziehen (Macintosh). Unter Windows wählen Sie **Datei/Exportieren** und speichern die Klasse unter einem passenden Namen ab. Um die Klasse in einem anderen Projekt zu verwenden, draggen Sie sie vom Schreibtisch in das Projektfenster des neuen Projekts.

Externe Projektelemente

Wenn Sie eine Klasse in mehreren Projekten verwenden möchten, können Sie diese Ihrem Projekt als externes Projektelement hinzufügen. Ein externes Projektelement wird auf der Festplatte gespeichert. Jedes Projekt, das davon Gebrauch macht, enthält einen Verweis darauf. Wenn die Klasse in einem Projekt eine Änderung erfährt, dann schlägt sich diese Veränderung auch in allen anderen Projekten, die auf das externe Element verweisen, nieder. Veränderungen, die Sie an einem externen Projektelement vornehmen, werden mit dem Speichern des Projekts gespeichert.

Um eine Klasse Ihres Projekts als externes Element zu speichern, machen Sie einen Ctrl-Klick auf die Klasse (Rechtsklick unter Windows) und wählen aus dem Kontextmenü „Externes Element erzeugen“, worauf ein „Datei exportieren“-Dialog erscheint. Nachdem Sie das Element gespeichert haben, wird es im Projektfenster kursiv dargestellt und das Icon erhält einen kleinen Pfeil.

Um einem anderen Projekt ein externes Element hinzuzufügen, halten Sie die Befehls (⌘)- und Wahltaste (z) gedrückt (Windows: Ctrl+Shift), während Sie das Element vom Schreibtisch auf das Projektfenster ziehen. Im Projektfenster wird der Name des externen Elements kursiv angezeigt.

Detaillierte Informationen zu diesem Thema finden Sie im Abschnitt „Externe Projektelemente“ auf Seite 83.

Verändern von Klassen

Ein großer Vorteil des Klassenkonzeptes ist die Möglichkeit, bestehende Klassen zu modifizieren. Dies tun Sie durch Hinzufügen von Konstanten, Eigenschaften und Änderungen an den Events oder den Methoden der Klasse.

Gültigkeitsbereich von Konstanten, Eigenschaften und Methoden einer Klasse

Wenn Sie einer Klasse eine Eigenschaft, Konstante oder Methode hinzufügen, müssen Sie deren Gültigkeitsbereich festlegen. Der Gültigkeitsbereich bestimmt, welche Teile des Projekts auf diese zugreifen können.

Öffentlich: Die Konstante, Eigenschaft oder Methode kann innerhalb des gesamten Projekts verwendet werden. Innerhalb des Objekts, in dem sie definiert wurde, erfolgt der Zugriff über den Namen, außerhalb über die Notation „Klassenname.Name“.

Geschützt: Geschützte Konstanten, Eigenschaften oder Methoden können nur in den Objekten der Klasse verwendet werden, in der sie erzeugt wurden. Andere Teile im Projekt können nicht darauf zugreifen.

Privat: Eine private Konstante, Eigenschaft oder Methode ist der geschützten sehr ähnlich. Allerdings kann dieses Element nur in Code verwendet werden, der direkt zur Klassse gehört. Es steht nicht in von der Klasse abgeleiteten Klassen zur Verfügung.

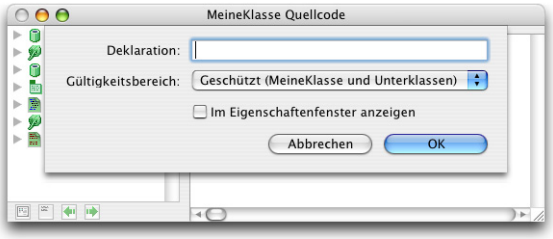
Hinzufügen von Eigenschaften

Sie können einer Klasse neue Eigenschaften zuordnen, um Werte zu speichern, die ihre Oberklasse nicht speichert. Beispielsweise könnte es sein, dass Sie einem EditField eine neue Eigenschaft geben möchten, damit es sich den letzten Wert merken kann, den der Benutzer eingegeben hat. Dies würde es Ihnen gestatten, auf Wunsch des Benutzers den letzten Wert des Feldes wieder herzustellen.

So versehen Sie eine Klasse mit einer neuen Eigenschaft:

1. Öffnen Sie den Code Editor der Klasse mit einem Doppelklick auf ihr Icon im Projektfenster.
2. Klicken Sie auf **Bearbeiten/Neue Eigenschaft**. Folgende Dialogbox erscheint:

Abb. 228: Der Eigenschaften-Deklarations-Dialog



3. Geben Sie den Namen der Eigenschaft, ein „As“ und den Datentyp der Eigenschaft an. Zum Beispiel *Name As String*.

   Die Eigenschaft kann auch ein Array sein. Ein String-Array mit vier Elementen, das einen Namen, Nachnamen, die Anschrift und eine Telefonnummer aufnimmt, würde man so formulieren:

   ```
   aNamen(3) As String
   ```

   Sie müssen die Größe des Arrays angeben, können sie aber später mit der ReDim-Methode verändern.
4. Wählen Sie einen Gültigkeitsbereich für die Eigenschaft und aktivieren Sie, wenn gewünscht, die Option **Im Eigenschaftenfenster anzeigen**.
5. Klicken Sie auf **OK**.

   Die Eigenschaft wird nun im Code-Editor angezeigt. Ist es eine geschützte oder private Eigenschaft, wird im Icon ein Warndreieck angezeigt.
6. Im Textfeld des Code-Editors können Sie Kommentare oder Notizen zu der Eigenschaft unterbringen.

Hinzufügen neuer Konstanten

Sie können einer Klasse auch Konstanten hinzufügen, denen Sie fixe Werte zuweisen. Eine Konstante verhält sich wie eine Eigenschaft, deren Wert man nicht verändern kann. Der Wert wird beim Erzeugen zugewiesen und ist dann unveränderlich.

Mit folgenden Schritten fügen Sie einer Klasse eine neue Konstante hinzu:

1. Öffnen Sie den Code-Editor der Klasse.
2. Rufen Sie den Menüpunkt **Bearbeiten/Neue Konstante** auf.

Daraufhin erscheint folgender Dialog:

3. Geben Sie den Namen, den Datentyp, den Gültigkeitsbereich und den Wert der Konstanten ein.

   Wenn Sie einen mehrzeiligen String eingeben wollen, klicken Sie auf den kleinen Button, neben dem Eingabefeld. Dann können Sie den String in einem eigenen Dialog eingeben.

4. Die Lokalisierungstabelle bietet Ihnen die Möglichkeit, der Konstanten abhängig von der beim Compilieren gewählten Sprache und Zielplattform verschiedene Werte zuzuweisen.

   Weitere Informationen finden Sie im Abschnitt „Lokalisieren einer Applikation mit Hilfe von Konstanten“ auf Seite 208.

5. Klicken Sie auf **OK**, um die Konstante zu speichern. Die Konstante erscheint daraufhin mit ihrem Namen, Wert und einem Icon, das ihrem Typ entspricht, im Code-Editor.

Hinzufügen neuer Methoden

Sie können neue Methoden zu einer Klasse hinzufügen, um diese Klasse mit zusätzlichen Funktionen zu versehen. Im GridLock-Beispiel gibt es beispielsweise eine Methode, die abhängig von der Anzahl an Zeilen und Spalten, die ihr übergeben wird, ein Gitter zeichnet.

Detaillierte Informationen zum Anlegen von Methoden finden Sie im Abschnitt „Methoden in einem Fenster anlegen“ auf Seite 209. Hier noch einmal das Vorgehen in Kurzform:

1. Führen Sie im Projektfenster einen Doppelklick auf die Klasse aus, um sie zu öffnen.
2. Wählen Sie **Bearbeiten/Neue Methode**. Die Dialogbox zum Anlegen einer neuen Methode erscheint.
3. Geben Sie einen Namen für die Methode ein.
4. Soll die Methode Parameter erhalten, so geben Sie diese durch Komma getrennt ein.
5. Falls es sich bei der Methode um eine Funktion handeln soll, legen Sie den Datentyp des Rückgabewertes fest.
6. Wählen Sie einen Gültigkeitsbereich für die Methode oder Funktion.
7. Klicken Sie auf **OK**, um die Methode oder Funktion zu speichern.

Der Code-Editor erscheint und zeigt die neu erstellte Methode/Funktion an.

Neue Events definieren

Events einer Klasse, zu denen Sie Programmcode definiert haben, sind einer Instanz dieser Klasse standardmäßig nicht zugänglich. Stellen Sie sich folgendes vor: Sie haben eine Klasse, die auf ListBox basiert. Deren Event-Handler „Open“ erweitern Sie um eigenen Code. Instanzen dieser Klasse werden keinen eigenen Event-Handler „Open“ haben. Der Hin-

tergedanke dabei ist, dass die Klasse den Code für das Event schon beinhaltet und daher auch für die Bearbeitung zuständig ist.

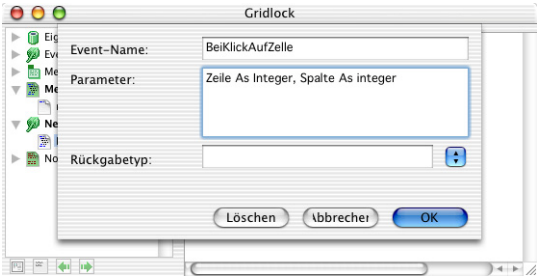
Es sind Fälle denkbar, in denen Sie jeder Instanz einer Klasse eigenen Code für einen Event mitgeben möchten. Dies gilt zum Beispiel für Ausgangswerte. Im Event-Handler „Open“ können Sie Ausgangswerte für die Klasse anlegen. Nehmen wir wieder das Beispiel der ListBox, die die Monatsnamen anzeigt und in der Sie den aktuellen Monat als Ausgangswert definieren wollen. Wird diese Klasse in einem Fenster verwendet, möchten Sie aber vielleicht diesen Ausgangswert ändern und einen anderen Monat als Vorgabe wählen. Die Instanz der ListBox wird aber keinen Event-Handler „Open“ haben und somit den Ausgangszustand erhalten, den die Klasse im Event „Open“ vorgibt.

Dieses Problem kann man mit neuen Events lösen. Legen Sie für die Klasse einen neuen „Open“-Event an und rufen Sie Sie ihn vom Open-Event-Handler der Klasse auf. Neue Events sind nur den Instanzen der Klassen zugänglich. Somit ist der neu definierte Event für jede Instanz der Klasse vorhanden. Er wird immer dann aufgerufen, wenn das Fenster geöffnet wird, so wie ein normaler „Open“-Event.

Ein weiteres Beispiel: Angenommen Sie definieren eine Klasse, die ein Gitter zeichnet. Das Gitter zeigt dem Benutzer einzelne Zellen, die dieser durch Anklicken ein- und ausschalten kann. Sie möchten nun einen neuen Event definieren, der immer dann eintritt, wenn der Benutzer eine Zelle anklickt. Nennen wir ihn „CellClicked“. Dieser Event soll die Information bekommen, in welche Zeile und Spalte geklickt wurde. Bei jeder Instanz der Klasse könnten Sie dann etwas auslösen, wenn der Benutzer in eine Zelle klickt.

Wie macht man das? Erzeugen Sie für die Klasse zunächst einen neuen Event und nennen Sie ihn „BeiKlickAufZelle“. Da Sie als Parameter noch Zeile und Spalte haben möchten, definieren Sie diese Parameter für den Event, so wie es die Abbildung zeigt.

Abb. 229: Ein neues Event anlegen



Nun stellt sich die Frage, wann der Event ausgelöst wird. Da der Benutzer mit der Maus klickt, um eine Zelle anzuwählen, sollte der Mausklick den Event auslösen. Die Gitter-Klasse würde man von der Klasse Canvas ableiten. Dort sind die Events „MouseDown“ und „MouseDrag“ für Mausklicks zuständig. Von dort rufen Sie **BeiKlickaufZelle** auf, als wenn es sich um eine Methode handeln würde. Sie berechnen die Zeile und Spalte und übergeben diese als Parameter an BeiKlickAufZelle.

Klickt der Benutzer auf eine Zelle, wird der Event „MouseDown“ der Klasse aktiviert. Dadurch wird auch „BeiKlickAufZelle“ aufgerufen und Zeile und Spalte werden übergeben. Der Event „BeiKlickAufZelle“ wird so für jede Instanz bereitgestellt. Die Klasse ruft eine Unterroutine der Instanz auf. Weil „BeiKlickAufZelle“ auch so angelegt werden kann, dass ein Wert zurückgegeben wird, kann die Instanz der Klasse Daten an die Oberklasse zurückliefern. Dies kann in unserem Beispiel dazu genutzt werden, um die Klicks zu filtern. Die Klasse würde den Klick immer dann weiter behandeln, wenn „BeiKlickAufZelle“ den Wert False liefert, weil das die Standardrückgabe einer Funktion ist. Das würde jeder Instanz sofort gestatten festzustellen, welche Zellen angeklickt werden können und welche nicht.

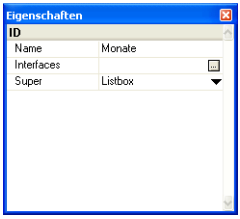
Ein Beispiel für diesen neuen Event finden Sie im „Gridlock“-Projekt auf der REALbasic CD.

Konstruktoren und Destruktoren

Manchmal ist es nützlich, wenn beim Erzeugen eines neuen Objekts automatisch eine Initialisierung des Objekts vorgenommen wird. Ein Konstruktor leistet genau diese Arbeit. Ein Konstruktor wird immer genau dann ausgeführt, wenn eine neue Instanz der Klasse erzeugt wird.

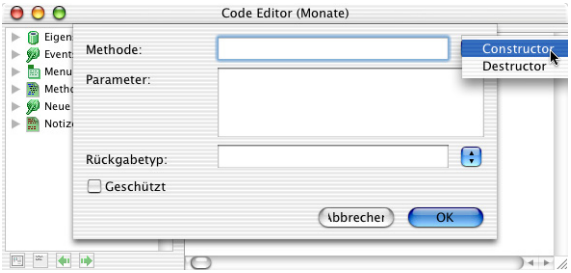
Angenommen, Sie brauchen eine ListBox, die als Grundeinstellung alle Monate anzeigt und bei der der aktuelle Monat die voreingestellte Auswahl ist. Legen Sie zunächst eine Unterklasse der ListBox mit dem Namen „Monate“ an. Wählen Sie dazu **Datei/Neue Klasse** und tragen Sie im Eigenschaftenfenster den Namen der Klasse ein und setzen Ihre Super(Class)-Eigenschaft auf ListBox.

Abb. 230: Eine Klasse mit ListBox als Super Class



Als nächstes klicken Sie im Projektfenster auf die „Monate“-Klasse und öffnen sie im Code-Editor. Wählen Sie **Bearbeiten/Neue Methode**. Jetzt können Sie einfach eine Konstruktor- oder Destruktor-Methode erzeugen. Anstatt der Methode einen Namen zu geben, wählen Sie dazu aus dem Popup rechts vom Namenseingabefeld entweder „Konstruktor“ oder „Destruktor“. Dann wird der korrekte Name für den Konstruktor bzw. Destruktor in das Namensfeld eingefügt. Da wir einen Konstruktor programmieren wollen, wählen Sie „Konstruktor“.

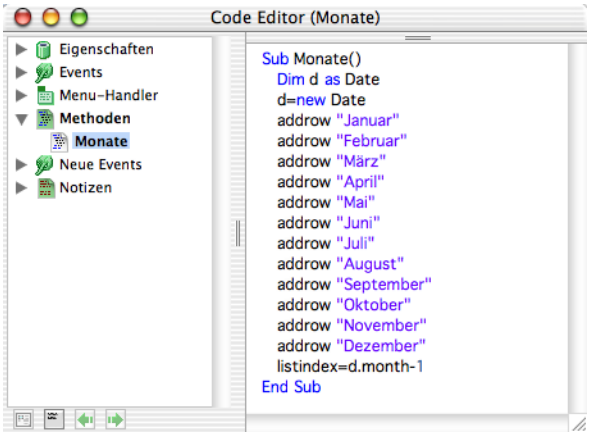
Abb. 231: Eine Konstruktor-Methode erzeugen



Als Methodenname wird der Name der Klasse eingefügt. Der Konstruktor der „Monate“-Klasse heißt also „Monate“.

Er verwendet folgenden Code:

Abb. 232: Der „Monate“-Konstruktor



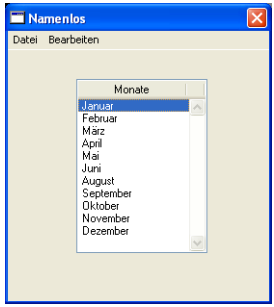
Beachten Sie, dass der Code nicht auf eine Instanz der ListBox verweist, da er der Unterklasse und nicht einer bestimmten Instanz, die sich von der ListBox-Klasse ableitet, hinzugefügt wird.

Fügen Sie dann einem Fenster eine Instanz der Monate-Klasse hinzu, indem Sie sie aus dem Projektfenster in den Fenstereditor draggen. In den Open-Event-Handler dieser Instanz schreiben Sie:

```
me.Monate
```

Der Konstruktor wird nun beim Starten der Anwendung aufgerufen. Als Ergebnis erhalten Sie eine ListBox, deren Inhalte automatisch eingetragen werden. Da im Konstruktor die ListIndex-Eigenschaft gesetzt ist, wird die aktuelle Zeile hervorgehoben.

Abb. 233: Die „Monate“-ListBox in der fertigen Applikation



Sie können auch einen Destruktor anlegen. Ein Destruktor wird automatisch ausgeführt, wenn eine Instanz der Elternklasse gelöscht bzw. nicht mehr benötigt wird (sich nicht mehr im Scope befindet). Zum Anlegen einer Destruktor-Methode wählen Sie „Destruktor“ aus dem Popup-Menü neben dem Namensfeld der „Neue Methode“-Dialogbox. Der Destruktor muss den gleichen Namen wie die Klasse mit vorangestellter Tilde („~“) tragen.

Overloading

REALbasic unterstützt auch das Overloading (Überladen) von Methoden. Eine Sprache, die Overloading unterstützt, erlaubt Ihnen, zwei oder mehr Methoden zu definieren, die den gleichen Namen besitzen, jedoch eine unterschiedliche Anzahl an Parametern bzw. Parameter unterschiedlicher Datentypen verwenden. Wenn der Methodenname aufgerufen wird, ermittelt REALbasic aus den Parametern, welche Methode gemeint ist. Ein gutes Beispiel für Overloading ist der

eingebaute „+“-Operator. Wenn seine Argumente Zahlen sind, berechnet er deren Summe, wenn die Argumente Strings sind, hängt er diese aneinander.

Doppelverwendung von selbstdefinierten Klassen

In REALbasic können Sie Operatoren „überladen“, um damit arithmetische und Vergleichsoperationen in Ihren selbst programmierten Klassen durchzuführen. Wenn Sie beispielsweise zwei Objekte, die Listen beinhalten, addieren wollen, können Sie eine Funktion schreiben, die den „+“-Operator für diese selbstdefinierte Klasse „überlädt“. Zu diesem Zwecke verwendet REALbasic eine Reihe von Schlüsselwörtern.

„Operator_*Schlüsselwort*“ steht für reservierte Wörter, die es Ihnen ermöglichen, die üblichen arithmetischen und Vergleichsoperatoren in Ihre selbstdefinierten Klassen zu integrieren.

Tabelle 34. Gültige Operatoren und Schlüsselwörter für selbstdefinierte Klassen

Operator	Operator_Schlüsselwort
+	Operator_Add Operator_AddRight
–	Operator_Subtract Operator_SubtractRight
*	Operator_Multiply Operator_MultiplyRight
/	Operator_Divide Operator_DivideRight
\	Operator_IntegerDivide Operator_IntegerDivideRight
Mod	Operator_Modulo Operator_ModuloRight
=,<,><=,>=	Operator_Compare
(Negation)	Operator_Negate
(Konvertierung)	Operator_Convert

Um beispielsweise zwei Instanzen einer selbstdefinierten Klasse addieren zu können, definieren Sie für diese Klasse eine Funktion mit dem Namen „Operator_Add“. Diese Funktion legt fest, wie Objekte dieser Klasse addiert werden. Dann können Sie einfach den „+“-Operator verwenden, um zwei Instanzen dieser Klasse zu addieren.

Weitere Information zu diesem Thema erhalten Sie im Abschnitt „Operator_Schlüsselwort“ in der Sprachreferenz.

Einer Methode einen Wert zuweisen

Wenn Sie eine Methode aufrufen, können Sie Ihr wahlweise einen Wert zuweisen. Dazu verwenden Sie die gleiche Syntax, die Sie bei der Zuweisung eines Wertes zu einer Eigenschaft benutzen würden. Sie schreiben also:

```
Objektname.Methodenname=Wert
```

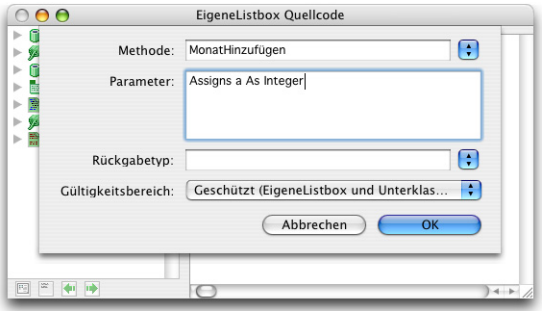
Dabei wird `Wert` dem Parameter zugewiesen, den die Methode als letzten erwartet. Der Wert muss dem Datentyp des Parameters entsprechen. Einer Methode, die mehrere Parameter erwartet, können Sie auf diese Weise nur den letzten Wert übergeben. Alle anderen Werte müssen wie üblich in Klammern hinter dem Methodennamen aufgezählt werden. Wenn Sie diese Syntax verwenden wollen, müssen Sie beim Deklarieren der Methode das Schlüsselwort „Assigns“ verwenden.

Beispiel: Sie haben eine Klasse „EigeneListbox“ definiert, die auf der ListBox-Klasse basiert. Für diese Klasse wollen Sie eine Methode programmieren, die abhängig von einem übergebenen Integer-Wert eine neue Zeile mit dem passenden Monatsnamen in die Listbox einfügt:

```
Select case a
 case 1
 addrow "Januar"
 case 2
 addrow "Februar"
 case 3
 addrow "März"
 case 4
 addrow "April"
 case 5
 addrow "Mai"
 case 6
 addrow "Juni"
 case 7
 addrow "Juli"
 case 8
 addrow "August"
 case 9
 addrow "September"
 case 10
 addrow "Oktober"
 case 11
 addrow "November"
 case 12
 addrow "Dezember"
end select
End Sub
```

Die Deklaration dieser Methode sieht folgendermaßen aus:

Abb. 234: Verwenden des „Assigns“-Schlüsselwortes beim Deklarieren einer Methode



Wenn Sie das Assigns-Schlüsselwort verwendet haben, können Sie den Integer-Wert mit dem Zuweisungsoperator an die Methode übergeben:

```
EigeneListbox.MonatHinzufügen=2
```

Nehmen wir einmal an, Sie schreiben für die „EigeneListBox“-Klasse eine weitere Methode namens „Tabellenfeld Ändern“, an die Sie die Parameter „Zeile“ und „Wert“ übergeben können. „Zeile“ bezeichnet das Tabellenfeld der ListBox, das Sie ändern wollen, und „Wert“ bezeichnet den neuen Wert.

```
Sub TabellenfeldÄndern(Zeile as Integer, Assigns Wert as String)
Cell(Zeile,0)=Wert
```

```
End Sub
```

Falls Sie den dritten Eintrag der ListBox auf „New York“ ändern möchten, würde der herkömmliche Aufruf so aussehen:

```
EigeneListBox.TabellenfeldÄndern(2,"New York")
```

Bei der Verwendung von Assigns können Sie die Zeile als Parameter und den neuen Wert mittels Zuweisungsoperator angeben:

```
EigeneListBox.TabellenfeldÄndern(2)="New York"
```

Klassen-Arrays

Genau wie mit Objekten ist es möglich, ein Array aus Instanzen einer Klasse zu erzeugen. Beispielsweise können Sie ein Array aus Steuerelementen der Benutzeroberfläche Ihres Programms erzeugen, um die Verwaltung der Benutzeroberfläche zu vereinfachen. Dazu geben Sie allen Steuerelementen den gleichen Namen und unterscheiden sie anhand ihrer Index-Eigenschaft. Arrays aus Steuerelementen wurden im Abschnitt „Gemeinsamer Code für ein Array aus Steuerelementen“ auf Seite 219 vorgestellt.

Es ist möglich, ein Array aus Instanzen einer Klasse zu erzeugen, die kein Steuerelement ist. Dabei können Sie die Objekthierarchie oder Klassen-Interfaces ausnutzen. Wenn myClass1 von mySuperClass1 oder von einem von myClass1 implementierten Klassen-Interface abgeleitet wurde, kann ein Array von myClass1 überall verwendet werden, wo ein Array von mySuperClass1 erwartet wird. Das Array behält seine ursprünglichen Elementtypen bei. Jedes Einfügen, Ändern und jede Zuweisung an Array-Elemente wird überprüft, um sicher zu stellen, dass der neue Wert den richtigen Elementtyp hat. Wenn der Typ nicht passt, wird eine TypeMismatchException ausgelöst. Diese können Sie mit einer Catch-Anweisung in einem Try-Block oder in einem Exception-Block abfangen.

Casting

Eine leistungsfähige Möglichkeit, wiederverwendbaren Code zu schreiben, besteht darin, die Objekthierarchie mittels *Casting* auszunutzen. Am besten lässt sich dies mit einem Beispiel illustrieren:

Da von Ihnen erzeugte Objekte Unterklassen der Basisklassen der REALbasic-Sprache sind, können Sie mit dem *IsA*-Operator testen, ob ein Objekt Mitglied einer bestimmten Unterklasse ist. Ist dies der Fall, können Sie das Objekt auf diesen Typ „casten“ und Operationen mit dem Objekt ausführen.

Das Beispiel verwendet eine For-Schleife, die alle Steuerelemente eines Fensters durchläuft und jedes darauf testet, ob es ein EditField ist. Ist dies der Fall, wird das Steuerelement auf EditField gecastet und die Werte seiner Text- und DataField-Eigenschaften ermittelt. Der Text wird dann einem Feld eines Datensatzes zugewiesen, das so heißt wie die DataField-Eigenschaft des EditFields.

```
Dim r as DatabaseRecord
Dim fieldname, fieldContents as String
.
.
For i = 1 to Self.ControlCount  //Anzahl Steuerelemente im Fenster
  If Self.control(i) IsA EditField then
    fieldname = EditField(control(i)).DataField //auf EditField casten
    //Die Text-Eigenschaft, die dem Inhalt dieses Feldes zugewiesen ist
    fieldContents = EditField(control(i)).text
    r.column(fieldname) = fieldContents
  end if
next
```

Dieser Code ist universell, da er auf keine spezifischen Fenster, EditFields oder Datenbanken verweist. Daher können Sie diese Routine in beliebigen Projekten für Datenbank-Eingabemasken verwenden.

Klassen kontrollieren Menüpunkte

Sobald eine Klasse den Fokus erhält, hat sie die Kontrolle über die Menüleiste. Dies erleichtert das Kapseln von Code in einem Steuerelement. Unterklassen von Klassen, die den Fokus erhalten können, verfügen über einen EnableMenuItems-Event-Handler und können für jedes Menü des Projekts einen eigenen Event-Handler haben.

Der EnableMenuItems-Event-Handler einer Klasse wird immer dann ausgelöst, wenn der Anwender in die Menüleiste klickt oder einen Tastatur-Shortcut für einen Menüpunkt tippt. Dadurch hat die Klasse die Gelegenheit, Menüpunkte ein- oder auszuschalten. Als nächstes wird der EnableMenuItems-Event-Handler des Fensters, gefolgt von dem der Application-Klasse ausgelöst. Wird dann ein Menüpunkt ausgewählt, prüft REALbasic, ob die Klasse, bei der gerade der Fokus liegt, einen Menu-Handler für den gewählten Menüpunkt hat. Ist dieser vorhanden, so wird er ausgeführt, gefolgt vom Menu-Handler des Fensters und von dem der Application-Klasse (jeweils unter der Annahme, dass diese über einen Menu-Handler für den ausgewählten Menüpunkt verfügen).

Sie erinnern sich vielleicht an das SecureEditField, das im Abschnitt „Beispiele von Unterklassen“ auf Seite 306 vorgestellt wurde. Dies ist ein Beispiel für eine Klasse, die Menüpunkte kontrolliert. Dort wurde schon erläutert, wie Menüpunkte ausgeschaltet werden, die REALbasic sonst automatisch aktiviert.

Verwendung von Klassen im eigenen Projekt

Bevor Sie in Ihren Projekten Klassen verwenden können, sollten Sie sich mit den zugrundeliegenden Konzepten und Begriffen vertraut machen. Dabei sind drei Begriffe von besonderer Bedeutung: Klasse, Instanz und Referenz.

Die Klasse

Eine Klasse ist eine Vorlage für eine Sammlung aus Events, Methoden und Eigenschaften, aus der Sie Instanzen und Unterklassen erzeugen können.

Die Instanz

Jede Instanz ist ein Speicherbereich, in dem eine Kopie der Eigenschaften einer Klasse gespeichert wird. Methoden werden nicht mit jeder Instanz in den Speicher kopiert, sondern erst dann aus der Klasse in den Speicher geladen, wenn sie aufgerufen werden.

Die Referenz

Die Referenz ist ein Wert, der in einer Eigenschaft oder einer lokalen Variablen abgelegt wird und auf die Instanz einer Klasse im Speicher verweist. Sie verwenden diese Eigenschaft oder Variable, um auf die Instanz der Klasse zuzugreifen. Hier ein Beispiel:

```
Dim person as Programmer
person=New Programmer
person.name="Jason"
```

„person“ ist eine lokale Variable, die eine Referenz auf eine Instanz der Klasse „Programmer“ speichert. Dazu wird sie als „Programmer“ definiert. Mit dem Kommando „New“ wird eine neue Instanz der Klasse „Programmer“ im Speicher angelegt. Sie müssen sich das so vorstellen, dass jeder „New“-Befehl eine neue Kopie der Klasse nach der durch die Klasse definierten Vorlage im Speicher anlegt.

Die Zuweisung beim Kommando „New“ bewirkt nun, dass eine Referenz auf die neue Instanz in der lokalen Variable „person“ abgelegt wird. REALbasic kann nun über den Wert in „person“ den Speicherbereich wiederfinden, in dem sich die Instanz befindet. In der nächsten Zeile des Beispiels wird die Referenz dazu benutzt, in der Eigenschaft „Name“ den Namen „Jason“ abzulegen.

Auf Steuerelementen basierende Unterklassen

Die einfachste Möglichkeit einem Projekt eine Klasse, die auf einem Steuerelement basiert, hinzuzufügen, besteht darin, das Steuerelement von der Steuerelementepalette auf das Projektfenster zu ziehen. Daraufhin wird dem Projekt eine neue Klasse mit dem Namen „SteuerelementnameX" hinzugefügt und das Eigenschaftenfenster zeigt an, dass die Klasse eine Unterklasse der Steuerelement-Klasse ist. Das „X" wird dabei durch eine fortlaufende Zahl ersetzt. Verwenden Sie das Eigenschaftenfenster, um der Klasse einen neuen Namen zu geben.

Sie eine Unterklasse eines Steuerelements auch dadurch anlegen, dass Sie **Datei/Neue Klasse** wählen. Damit legen Sie eine neutrale Klasse an. Im Eigenschaftenfenster können Sie die Super(Class)-Eigenschaft auf die Klasse eines Steuerelements ändern.

Da Sie die Klasse von einem Steuerelement abgeleitet haben, können Sie der Klasse neue Eigenschaften und Methoden hinzufügen, um ihr individuelles Steuerelement zu erzeugen und anschließend eine Instanz ihres individuellen Steuerelements einem Fenster hinzufügen.

Um eine Instanz einer Klasse zu erzeugen, der ein Steuerelement zugrunde liegt, ziehen Sie einfach das Element vom Projektfenster in das Fenster, in dem Sie die neue Instanz verwenden wollen – gerade so, als ob Sie eine Instanz eines eingebauten Steuerelements erzeugen wollten, indem Sie sein Icon aus der Steuerelementepalette „herausdraggen".

Andere Instanzen/Klassen

Klassen müssen nicht auf Steuerelementen aufbauen, sondern können auch von Klassen abgeleitet werden, die nicht Bestandteil der Control-Klasse sind. Ein Beispiel dafür wäre die Thread-Klasse. Es ist möglich, eine Unterklasse der Thread-Klasse zu erzeugen und für diese neue Eigenschaften zu definieren, in denen bestimmte Informationen abgelegt werden können.

Es ist auch möglich, eine ganz neue Klasse anzulegen, die gar keine Oberklasse besitzt. Beispielsweise könnten Sie eine Klasse „Personen" anlegen. Eigenschaften der Klasse wären dann Name, Alter und Größe. Von dieser Klasse könnten Sie wiederum eine Unterklasse „ComputerAnwender" ableiten. Diese könnte weitere Eigenschaften definieren, die Sie für diese Gruppe von Personen speichern wollen.

Um eine Instanz auf eine solche Klasse zu erzeugen, müssen Sie zunächst etwas haben, um diese zu speichern. Sie können diese in einer Eigenschaft oder in einer lokalen Variablen ablegen. Diese müssen vom Typ der Klasse oder einer ihrer Oberklassen sein. Benötigen Sie beispielsweise eine Instanz der Date-Klasse, um ein Geburtsdatum zu speichern, erzeugen Sie eine Variable vom Typ Date. Dort legen Sie dann eine Referenz auf die erzeugte Instanz ab.

Mit der Dim-Anweisung erzeugen Sie die Variable. Mit dem New-Operator erzeugen Sie eine neue Instanz und speichern in der Variablen eine Referenz auf diese.

```
Dim Geburtstag as Date
Geburtstag=New Date
```

In diesem Beispiel ist die lokale Variable „Geburtstag" vom Typ Date. Dann wird mit dem New-Operator eine neue Instanz der Date-Klasse erzeugt und eine Referenz in der lokalen Variable abgelegt.

Sie können die Schreibweise straffen, indem Sie den New-Operator mit der Dim-Anweisung verbinden. Damit erzeugen Sie in einem Schritt die Variable und die Instanz und weisen der Variablen die Referenz auf die Instanz zu.

```
Dim Geburtstag As New Date
```

Zugriff auf Methoden und Eigenschaften einer Klasse

Nachdem Sie eine Instanz einer Klasse und eine Referenz auf diese erzeugt haben, können Sie genau wie bei Objekten auf die Methoden und Eigenschaften der Klasse zugreifen. Im folgenden Beispiel hat die lokale Variable Geburtstag Zugriff auf die Eigenschaften der Date-Klasse. Sie können also mit Hilfe der lokalen Variable die Eigenschaften ändern.

```
Dim Geburtstag as New Date
Geburtstag.Year=1967
Geburtstag.Month=3
Geburtstag.Day=21
```

Dieses Beispiel basiert auf einer in REALbasic eingebauten Klasse. Wenn Sie eine abgeleitete Klasse verwenden, haben Sie natürlich Zugriff auf deren Eigenschaften und zusätzlich auf die Eigenschaften der Klasse, von der Sie Ihre Klasse abgeleitet haben.

Wann werden Instanzen von Klassen aus dem Speicher entfernt?

REALbasic verwaltet den Speicher selbst und verwendet dazu eine Technik namens Reference-Counting (Referenzzählung). REALbasic führt einen Zähler für jede Instanz, die sie erzeugen und zählt die Referenzen auf das Objekt. Jede Referenz erhöht den Zähler. Das Entfernen einer Referenz auf das Objekt verkleinert den Zähler. Sie entfernen eine Referenz, indem Sie sie z.B. auf Nil setzen oder das Fenster schließen, in der sich die Instanz befand. Wenn der Reference-Counter den Wert Null annimmt, wird das Objekt sofort aus dem Speicher entfernt. Instanzen von Klassen werden also automatisch aus dem Speicher entfernt, sobald sie nicht mehr gebraucht werden.

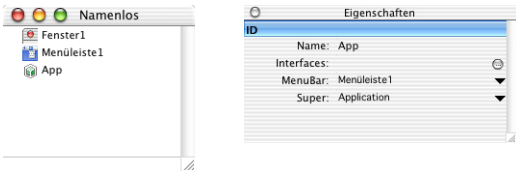
Angenommen, Sie haben in einem Fenster eine Instanz einer ListBox angelegt. Wenn das Fenster geöffnet wird, wird im Speicher automatisch eine Instanz der ListBox erzeugt. Beim Schließen des Fensters wird diese wiederum automatisch aus dem Speicher entfernt. Wird eine Referenz auf eine Klasse in einer lokalen Variablen abgelegt, so verschwindet die damit erzeugte Instanz wieder aus dem Speicher, sobald das Event oder die Methode, die die Instanz erzeugt hatte, beendet wird. Ist eine Referenz auf eine Instanz in einer Eigenschaft einer Klasse gespeichert, wird der Speicher dafür freigegeben, sobald das Objekt, zu dem diese Eigenschaft gehört, aus dem Speicher entfernt wird.

Die Application-Klasse

Aus der Application-Klasse können Sie eine Klasse ableiten, die Ihre Anwendung repräsentiert. Deshalb kann es in einem Projekt nur eine Klasse geben, die auf der Application-Klasse basiert. Beim Anlegen eines neuen „Desktop-Application"-Projekts wird diesem automatisch eine Unterklasse der Application-Klasse, die App-Klasse, hinzugefügt.

Das Eigenschaftenfenster der App-Klasse zeigt „Application" als „Super"-Klasse. Außerdem gehört auch die Standard-Menüleiste (Menüleiste1) der App-Klasse.

Abb. 235: Die Standard-Applikation-Klasse



Spezielle Event-Handler der Application-Klasse

Die Application-Klasse bietet spezielle Event-Handler. Diese sind:

- *Open*: Wird ausgeführt, wenn das Programm mit **Debug/Programm starten** (⌘-R oder Strg-R) oder als Stand-Alone-Programm gestartet wird.
- *Close*: Wird ausgeführt, wenn das Programm beendet wird.
- *NewDocument*: Wird ausgeführt, wenn die Applikation als Stand-Alone-Programm gestartet wird. Wird nicht ausgeführt, wenn die Applikation durch Doppelklick auf ein von ihr erzeugtes Dokument gestartet wird.

- *OpenDocument*: Wird ausgeführt, wenn die Applikation als selbständiges Programm durch Doppelklick auf eine dem Programm zugeordnete Dokumentendatei gestartet wird.
- *EnableMenuItems*: Wird ausgeführt, wenn der Benutzer in die Menüleiste klickt, und zwar bevor das Menü gezeichnet wird. Vorher werden die EnableMenuItems-Event-Handler aller Klassen mit Instanzen im obersten Fenster und der EnableMenuItems-Event-Handler des obersten Fensters selbst ausgeführt. Daher sollte man den EnableMenuItems-Event-Handler der Application-Klasse zur Aktivierung der Menüpunkte verwenden, die nicht davon abhängen, ob ein bestimmtes Fenster geöffnet ist oder nicht.

  Soll ein Menüpunkt immer aktiviert sein, verwenden Sie für diesen die AutoEnable-Eigenschaft und nicht den Event-Handler.
- *HandleAppleEvent*: Wird ausgeführt, wenn die Anwendung einen AppleEvent empfängt.
- *Activate*: Tritt auf, wenn die Anwendung geöffnet oder nach vorne geholt wird.
- *Deactivate*: Die Anwendung wird deaktiviert. Tritt auf, wenn eine andere Anwendung oder der Schreibtisch in den Vordergrund rückt oder wenn die Anwendung beendet wird.
- *Unhandled Exception*: Wird ausgeführt, wenn ein Laufzeitfehler auftritt, der nicht von einem ExceptionBlock abgefangen wird. Dieser Event bietet Ihnen eine „letzte Chance", Laufzeitfehler abzufangen, bevor diese Ihre Anwendung „unerwartet" beenden.

Eigenschaften der Application-Klasse sind global

Öffentliche Eigenschaften der Application-Klasse sind global verfügbar. Sie können also von jedem Teil des Projekts verwendet werden. Stellen Sie dazu den Gültigkeitsbereich der Eigenschaft auf „Öffentlich". Die abgebildete Eigenschaft der App-Klasse kann von jedem Teil des Projekts verwendet werden:

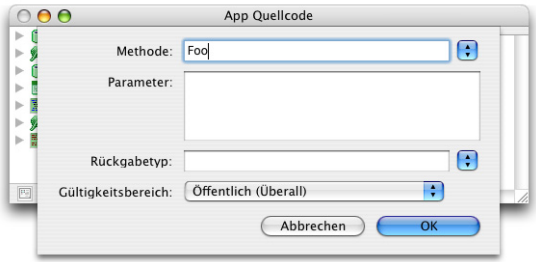
Verwenden Sie die App-Funktion, um außerhalb der App-Klasse auf die Eigenschaften der App-Klasse zuzugreifen. Um z.B. von einem anderen Teil des Projekts auf die Fehlercode-Eigenschaft zuzugreifen, verwenden Sie folgende Syntax:

```
App.Fehlercode
```

Ist der Gültigkeitsbereich „Geschützt", kann die Eigenschaft nur innerhalb der App-Klasse verwendet werden.

Methoden der Application-Klasse sind global

Auf öffentliche Methoden der Application-Klasse kann überall im Projekt zugegriffen werden. Dazu muss als Gültigkeitsbereich „Öffentlich“ gewählt werden. Wenn Sie der App-Klasse eine Methode namens „Foo“ hinzufügen,



können Sie diese folgendermaßen aufrufen:

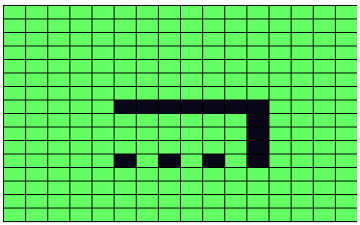
```
App.Foo
```

Ist der Gültigkeitsbereich „Geschützt“ oder „Privat“, so ist die Methode nur innerhalb der App-Klasse erreichbar.

Anlegen selbstdefinierter Steuerelemente mit Klassen

Eine der wichtigsten Anwendungen der Klassen ist die Definition eigener Steuerelemente. Die meisten Steuerelemente sind in REALbasic zwar schon vorhanden, aber es wird dennoch Fälle geben, in denen Sie selbst solche Elemente definieren möchten. Angenommen, Sie möchten ein Gitter mit Zellen erzeugen. Der Benutzer soll auf Zellen klicken können, um diese zu selektieren. Die Abbildung zeigt ein Beispiel, wie das aussehen könnte.

Abb. 236: Selbstdefiniertes Grid-Steuerelement



Selbstdefinierte Steuerelemente sind Unterklassen und werden von der Canvas-Klasse abgeleitet. Die Canvas-Klasse stellt eine Zeichenfläche zur Verfügung, auf der das Steuerelement dargestellt werden kann und empfängt alle nötigen Events, über die Anwender und Steuerelement interagieren können.

Im Beispiel des Grid-Steuerelementes in Abbildung 236 ist das Objekt eine Instanz der GridLock-Klasse, die von der Canvas-Klasse abgeleitet wurde. Das Beispielprogramm „Gridlock“ finden Sie auf der REALbasic CD. Die GridLock-Klasse übernimmt alle Eigenschaften und Events des Canvas-Steuerelements und besitzt eigene Methoden, Eigenschaften und Events zum Zeichnen und zur Interaktion. Das Gitter wird im Paint-Event-Handler des GridLock-Steuerelements gezeichnet.

In Eigenschaften der „Gridlock“-Klasse wird die Zeilen- und Spaltenzahl gespeichert. Diese können für jede Instanz verwendet werden. Weitere Eigenschaften speichern die Farbe für selektierte und nicht selektierte Zellen. Klickt der Benutzer auf eine Zelle, wird der Event-Handler „MouseDown“ der Gridlock-Klasse aktiv. Der Code dieses Event-Handlers stellt fest, welche Zelle angeklickt wurde und ob diese selektiert oder deselektiert werden muss. Zur GridLock-Klasse gehört ein Event „CellClicked“, das aufgerufen wird, wenn der Benutzer auf eine Zelle klickt. Damit kann eine Instanz der Gridlock-Klasse auf einen Klick reagieren. „CellClicked“ erhält als Parameter die Zeilen- und Spaltennummer der

angeklickten Zelle und ist als Funktion definiert. Wenn eine Instanz der Gridlock-Klasse im CellClicked-Event-Handler den Wert True zurückliefert, geht die Gridlock-Klasse davon aus, dass der Programmierer den Klick herausfiltern wollte und verhält sich so, als ob der Benutzer die Zelle nicht angeklickt hätte.

Zeichnen eines Steuerelements

Der „Paint"-Event-Handler der Canvas-Klasse (oder einer von ihr abgeleiteten Klasse) wird immer dann aufgerufen, wenn das Steuerelement neu gezeichnet werden muss. Dies passiert zum Beispiel, wenn ein Fenster das Steuerelement verdeckt hatte und nun geschlossen wird. Verändert sich das Aussehen eines Elements während der Benutzung nie, kann das Zeichnen des Elements immer komplett vom Paint-Event ausgeführt werden. Müssen Änderungen eingeplant werden, ist anders vorzugehen. Das Gridlock beispielsweise ändert sich immer, wenn der Benutzer eine Zelle anklickt. Außerdem hat es eine Funktion (DefineGrid), die es erlaubt, permanent die Anzahl der Zeilen und Spalten zu verändern. Solche Aktionen erfordern natürlich ein Neuzeichnen des Elements.

Der Code für das Neuzeichnen des Elements findet sich normalerweise in einer eigenen Methode. Im Gridlock-Beispiel heißt diese „DrawGrid". Sie bekommt die Graphics-Eigenschaft der Canvas-Klasse geliefert, die sie verwendet, um das Gitter neu zu zeichnen. Dadurch, dass der Code dafür in einer eigenen Methode liegt, kann diese bequem sowohl vom Event-Handler „Paint" als auch von der DefineGrid-Methode aufgerufen werden. Dem Paint-Event-Handler wird eine Referenz auf die Graphics-Eigenschaft übergeben, so dass diese an die DrawGrid-Methode weitergegeben werden kann, wenn sie vom Paint-Event-Handler aufgerufen wird. Die Methode „DefineGrid" ruft die Methode „DrawGrid" auf, da bei einer Änderung der Größe logischerweise auch das Gitter neu gezeichnet werden muss. Die Methode „DrawGrid" erhält die Graphics-Eigenschaft in folgender Syntax:

```
DrawGrid Me.Graphics
```

„Me" ist dabei eine Referenz auf die Instanz der Gridlock-Klasse. Obwohl der Code in diesem Fall innerhalb der Gridlock-Klasse aufgerufen wird, gestattet die Benutzung der Referenz „Me" den Zugriff auf die Eigenschaften der aktuell im Gebrauch befindlichen Instanz.

Virtuelle Methoden

Virtuelle Methoden erlauben es, dass eine Unterklasse eine eigene Version einer Methode verwendet. Normalerweise erben Unterklassen die Methoden ihrer Elternklasse.

Das Konzept virtueller Methoden wird auch als „Polymorphismus" bezeichnet.

Wenn eine Unterklasse eine Methode besitzt, die denselben Namen wie eine Methode der Elternklasse hat, wird immer die Methode der Unterklasse ausgeführt, es sei denn, man verwendet die Syntax:

```
elternklassenname.methodenname
```

So wird eine virtuelle Methode erzeugt:

1. Erzeugen Sie eine Klasse.
2. Fügen Sie der Klasse eine Methode hinzu.
3. Erzeugen Sie eine Unterklasse dieser Klasse.
4. Fügen Sie der Unterklasse eine Methode mit dem Namen der Klasse aus Schritt 2 hinzu.

Wenn die Unterklasse die Methode aufruft, wird sie ihre eigene Version, nicht die der Elternklasse verwenden.

Klasseninterface

Ein Klasseninterface ist ein Konstrukt, mit dem Sie verschiedene Klassen zusammenfassen können, die eine gleichartige Funktion erfüllen. Verstehen Sie ein Klasseninterface als eine Art Spezifikation, die verschiedene Methoden definiert. Ein Klasseninterface enthält allerdings nur die Definition, nicht den Code selbst.

Eine Methode in einem Klasseninterface ist tatsächlich nur ein Platzhalter für Methoden, die letztendlich in einer beliebigen Klasse implementiert sind. In dem Fall spricht man davon, dass die Klasse das Klasseninterface implementiert. Dabei müssen alle im Klasseninterface spezifizierten Methoden implementiert werden. Die Deklarationen müssen übereinstimmen, aber der Inhalt der Methoden kann unterschiedlich sein. Dies wird manchmal auch Polymorphismus genannt. Beispielsweise wird eine Methode, die den Text eines StaticText-Objekts ändert, anders formuliert sein, als eine, die den Text in einem Canvas-Objekt mit Hilfe des Graphics-Objekts ändert.

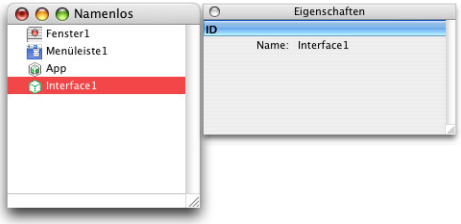
Der gesamte Prozess lässt sich in drei Phasen aufteilen:

- Erzeugen eines Klasseninterfaces
- Erzeugen der Klassen, die das Klasseninterface implementieren
- Hinzufügen der Klassen zum Projekt und Verwenden der Klassen. Üblicherweise bedeutet dies, dass Ihr Programm prüft, ob ein Klasseninterface implementiert ist und es dann verwendet.

Um ein Klasseninterface zu erzeugen, führen Sie folgende Schritte aus:

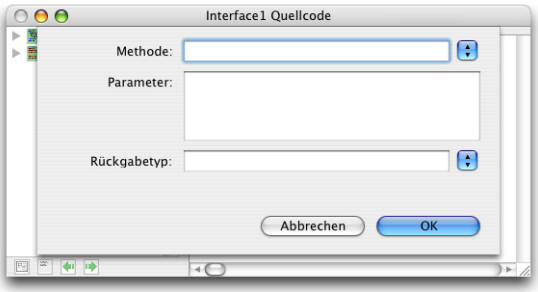
1. Wählen Sie den Menüpunkt **Ablage/Neues Klasseninterface**.

   Ein neues Klasseninterface wird dem Projekt hinzugefügt und das Icon erscheint leicht durchsichtig im Projektfenster, was signalisieren soll, dass dort kein Code implementiert ist. Das Eigenschaftenfenster zeigt, dass die einzige Eigenschaft eines Klasseninterfaces sein Name ist. Es hat auch keine übergeordnete Klasse, da es außerhalb der Objekt-Hierarchie liegt.



2. Jetzt können Sie den Namen des Klasseninterfaces im Eigenschaftenfenster ändern.
3. Klicken Sie im Projektfenster doppelt auf das Klasseninterface, um dessen Code-Editor zu öffnen.

   Der Browser des Code-Editors enthält nur die Einträge „Methoden“ und „Notizen“. Sie können keine Konstanten oder Eigenschaften für ein Klasseninterface erzeugen.
4. Rufen Sie **Bearbeiten/Neue Methode** auf, um eine Methode für das Klasseninterface zu deklarieren.



5. Bestimmen Sie den Namen, die Parameter und den Rückgabetyp, falls ein Rückgabewert benötigt wird.

   Anders gesagt: Deklarieren Sie die Methode, wie Sie es bereits gelernt haben. Nur einen Gültigkeitsbereich können Sie nicht angeben, da das Klasseninterface selbst keinen Code enthält. Es dient ja lediglich dazu, die Methoden zu spezifizieren, die anderswo implementiert werden.

   Wenn Sie die Deklaration mit Klick auf **OK** abgeschlossen haben, werden Sie feststellen, dass der Code-Editor es nicht gestattet, Code für diese Methode einzugeben. Dies ist so beabsichtigt und kein Fehler.

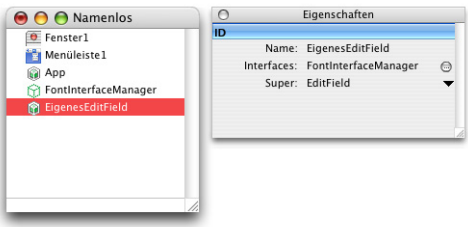
6. Wiederholen Sie die Schritte 4 und 5, bis alle benötigten Methoden des Klasseninterfaces deklariert sind.

   Nachdem Sie das Klasseninterface erzeugt haben, müssen Sie es einer oder mehreren abgeleiteten Klassen zuweisen. Um das Beispiel nicht zu einfach zu halten, gehen wir von zwei oder mehr Klassen aus. Nun müssen Sie in jeder der zugewiesenen Klassen die deklarierten Methoden implementieren.

Um ein Klasseninterface zu implementieren gehen Sie folgendermassen vor:

1. Bei aktivem Projektfenster wählen Sie den Menüpunkt **Ablage/Neue Klasse**.

   Eine neue Klasse wird dem Projekt hinzugefügt. Das Eigenschaftenfenster der Klasse enthält die Eigenschaft „Interfaces“, der Sie ein Klasseninterface zuweisen können. Sie können auch mehrere Klasseninterfaces angeben.



2. Geben Sie den Namen des Klasseninterfaces an, das von der Klasse implementiert wird.

   Eine Klasse kann mehrere Klasseninterfaces implementieren. Trennen Sie mehrere Interfaces mit Kommata.

3. Doppelklicken Sie die Klasse und öffnen Sie so ihren Code-Editor.
4. Wählen Sie **Bearbeiten/Neue Methode**.
5. Geben Sie die Deklaration an, die Sie für die Methode auch im Klasseninterface verwendet haben.

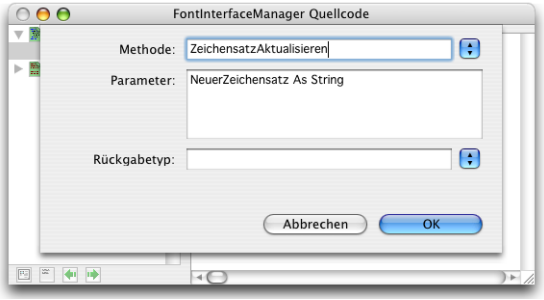
   Diesmal akzeptiert der Code-Editor die Eingabe von Code.

6. Schreiben Sie die Implementierung der Methode für diese Klasse.
7. Wiederholen Sie die Schritte 4 bis 6 für die Methoden jedes weiteren Klasseninterfaces, das diese Klasse implementiert.
8. Wiederholen Sie den gesamten Vorgang für jede Klasse, die ein Klasseninterface implementiert.

Beispielprojekt eines Klasseninterfaces

Das folgende Beispiel verdeutlicht das zugrundeliegende Konzept. Die Applikation besitzt zwei Klassen, die von den Klassen StaticText und EditField abgeleitet sind. Das Klasseninterface besteht aus einer Methode, die den Zeichensatz aktualisiert. Wenn der Benutzer einen Zeichensatz aus dem Menü auswählt, wird eine Nachricht an alle Steuerelemente geschickt, die das Klasseninterface implementieren.

Das Klasseninterface „FontInterfaceManager“ besitzt eine Methode:



Jede Klasse, die das Klasseninterface implementiert, muss eine Methode besitzen, die ZeichensatzAktualisieren heißt und einen String als Parameter akzeptiert.

Die zwei Klassen MeinStaticText und MeinEditField implementieren diese Methode, wie Sie den Eigenschaftenfenstern dieser Klassen entnehmen können:

Die Implementierung der Methode ZeichensatzAktualisieren ist zufällig in beiden Klassen identisch, dies muss aber nicht so sein!

```
Sub ZeichensatzAktualisieren(s As String)
  Self.TextFont=s
End Sub
```

Mit Objekten dieser Klasse in einem Fenster kann nun generischer Code geschrieben werden, der prüft, ob ein Steuerelement das Klasseninterface implementiert.

Nehmen wir an, es sind mehrere dieser Objekte in einem Fenster. Der folgende Code befindet sich im Change-Event eines PopupMenu-Objekts und ändert den Zeichensatz der Objekte, wenn der Benutzer einen neuen Zeichensatz auswählt. Mit Hilfe des IsA-Operators wird getestet, ob das Steuerelement das Klasseninterface implementiert. Trifft dies zu, wird dessen ZeichensatzAktualisieren-Methode aufgerufen. Über den Ausdruck Me.Text erreicht man den aktuell ausgewählten Eintrag des PopupMenus.

```
Dim i as Integer
For i=0 to Self.ControlCount  // Anzahl der Steuerelemente im Fenster
  If Self.Control(i) IsA FontInterfaceManager then
    FontInterfaceManager(Self.Control(i)).ZeichensatzAktualisieren(Me.Text)
 End if
Next
```

Jede selbst definierte Klasse kann also unabhängig von ihrer Superklasse ein Klasseninterface implementieren. Klasseninterfaces erlauben es, Klassen miteinander in Beziehung zu setzen, die in der Objekthierarchie in keinerlei Beziehung stehen, und Methoden für diese Klassen an einer zentralen Stelle des Projekts zu verwalten.

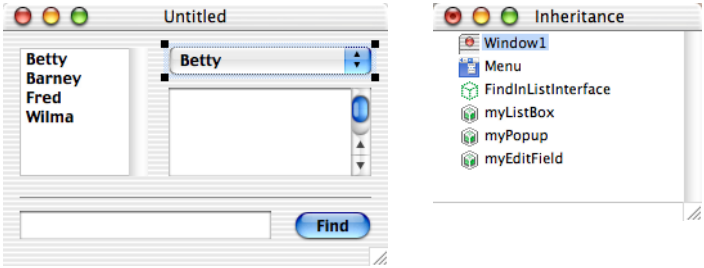
Interface-Vererbung

Interface-Vererbung dient zur komfortablen Lösung eines häufig auftretenden Problems. Wenn Ihre Anwendung mehrere verschiedene Steuerelemente verwendet, die dieselbe Aufgabe auf unterschiedliche Art und Weise ausführen sollen (abhängig vom konkreten Typ jedes Steuerelements), können Sie dank Interface-Vererbung den Interface-spezifischen Teil Ihres Programmcodes auf elegante Art und Weise schreiben und ausführen.

Abbildung 237 zeigt eine Applikation, die Interface-Vererbung verwendet. Die Applikation soll einen vom Benutzer eingegebenen String in den drei Steuerelementen über der Separator-Linie finden: Das EditField, die ListBox und das PopupMenu basieren jeweils auf benutzerdefinierten Klassen. Obwohl die Aufgabe für die drei Steuerelemente identisch ist (eine Suchfunktion), kann sie nicht mit exakt dem gleichen Code realisiert werden, da die drei Objekte ihre Daten unterschiedlich speichern und manipulieren. Deshalb verwendet jede benutzerdefinierte Klasse ihre eigene Implementierung der zum Suchen notwendigen Methoden.

Die ListBox, das EditField und das PopupMenu wurden von benutzerdefinierten Klassen abgeleitet, die alle ein benutzerdefiniertes Interface, das FindInListInterface, verwenden. Alle drei Steuerelemente verfügen über eine Suchen-Funktion, die dieselben Parameter empfängt, aber unterschiedlich implementiert ist. Der Programmcode für den Find-Button kann die verschiedenen Suchen-Funktionen mit derselben Syntax aufrufen.

Abb. 237: Eine Beispielanwendung für Interface-Vererbung



Der Anwender gibt den zu suchenden String im EditField FindValue links neben dem Find-Button ein. Wenn der Anwender den Find-Button anklickt, wird folgender Code ausgeführt:

```
FindIt FindValue.text, listBox1
FindIt FindValue.text, popupMenu1
FindIt FindValue.text, editField1
```

Die Funktion FindIt wird für alle drei Steuerelemente aufgerufen. Tatsächlich wird in jeder Programmzeile eine andere Version von FindIt ausgeführt, und zwar die zum jeweiligen Steuerelement passende. Der zweite Parameter von FindIt ist der Name des jeweiligen Steuerelements. Jedes Steuerelement hat Methoden von der benutzerdefinierten Klasse geerbt, auf der es basiert.

Das EditField, das PopupMenu und die ListBox sind Instanzen von benutzerdefinierten Klassen. Diese Klassen verfügen über die zwei Methoden Find und SelectRow, die die Suchfunktion für das jeweilige Objekt implementieren (siehe Tabelle 35).

Tabelle 35. Finden-Funktionen und SelectRow-Methoden für verschiedene Steuerelemente

Steuerelement	Find Funktion	SelectRow Methode

EditField	Function Find (FindValue as string) as Integer
	dim rows, foundPos, foundCRpos as integer
	rows=-1
	foundPos=instr(text,findValue+chr(13))
	do until foundCRpos>=foundPos
	foundCRpos=instr(foundCRpos+1, text, chr(13))
	rows=rows+1
	loop
	return rows
		Sub SelectRow (Row as Integer)
		dim counter, startPos, endPos as integer
		do until counter=row
		startPos=instr(startPos+1, text, chr(13))
		if startPos <> 0 then
		counter=counter+1
		end if
		loop
		endPos=instr(startPos+1,text,chr(13))
		selStart=startPos
		selLength=endPos-startPos

ListBox	Function Find (FindValue as string) as Integer
	dim i as integer
	for i=0 to listcount-1
	if list(i)=findValue then
	return i
	end if
	next
		Sub SelectRow (Row as Integer)
		listindex=row

PopupMenu	Function Find (FindValue as string) as Integer
	dim i as integer
	for i=0 to listcount-1
	if list(i)=findValue then
	return i
	end if
	next
		Sub SelectRow (Row as Integer)
		listindex=row

Die FindIt-Methode selbst verwendet das FindInListInterface:

```
Sub FindIt (findValue as String, source as SucheInListeInterface)
  dim row as integer
  source.selectRow source.find(findValue)
End Sub
```

Das Klasseninterface FindInListInterface deklariert lediglich die Methoden Find und SelectRow und deren Parameter. Wenn die FindIt-Methode aufgerufen wird, ruft sie immer die Versionen von Find und SelectRow auf, die zu dem Steuerelement gehören, das als zweiter Parameter an FindIt übergeben wurde.

Benutzerdefinierte Objektverknüpfungen

Mittels Objektverknüpfungen können Funktionen realisiert werden, ohne dass dafür Code geschrieben werden muss. In REALbasic sind bereits viele Aktionen eingebaut, die von verknüpften Objekten ausgeführt werden können (siehe auch „Object Binding – Objektverknüpfung“ auf Seite 135).

Über die vorhandenen Aktionen hinaus können Sie eigene Verknüpfungen programmieren, indem Sie benutzerdefinierte Klassen verwenden. Auch die selbstdefinierten Verknüpfungen erscheinen automatisch im Dialog „Neue Verknüpfung“, wenn Sie mit der Maus bei gedrückter Shift- und Befehlstaste (Windows: Shift- und Strg-Taste) eine Linie zwischen zwei geeigneten Objekten ziehen.

Eine selbstdefinierte Verknüpfung können Sie mittels Drag & Drop in jedes Projekt übernehmen. Sie wird genauso transparent funktionieren wie die in REALbasic eingebauten Verknüpfungen. Es folgen zwei Beispiele für benutzerdefinierte Objektverknüpfungen.

Die Verknüpfung „Lösche alle Zeilen“

Dieses erste Beispiel erzeugt eine benutzerdefinierte Verknüpfung zwischen PushButtons und ListBoxen. Mit ihr kann der Anwender einen PushButton anlegen, der alle Zeilen einer ListBox löscht. Als erstes erzeugen Sie eine benutzerdefinierte Klasse, die das Interface für die Verknüpfung enthält.

1. Fügen Sie Ihrem Projekt eine neue Klasse hinzu.
2. Nennen Sie die neue Klasse „DeleteAllRowsBind“.

   Da es sich bei dieser Klasse um eine Verknüpfung handeln soll, muss sie das bindingInterface unterstützen. Da außerdem im Action-Event des PushButtons eine Aktion ausgeführt werden soll, die der ListBox bei Klick auf den PushButton mitteilt, dass sie alle Einträge löschen soll, muss das ActionNotificationReceiver-Interface unterstützt werden.
3. Um das bindingInterface und das ActionNotificationReceiver-Interface zu unterstützen, tragen Sie „bindingInterface,ActionNotificationReceiver“ in die Interfaces-Eigenschaft der Klasse ein.

   Jetzt können Sie die bind-Methode implementieren.
4. Fügen Sie Ihrer Klasse die neue Methode „Bind“ mit den Parametern „Source as Object, Target as Object“ hinzu.

   Da der Code, der bei Klick auf den PushButton ausgeführt wird, wissen muss, welcher PushButton mit welcher ListBox verknüpft ist, müssen diese als Eigenschaften in der Klasse gespeichert werden.
5. Ergänzen Sie die Klasse um folgende neue Eigenschaften: BindSource as Pushbutton und BindTarget as ListBox.
6. Schreiben Sie folgenden Code in die Bind-Methode:

```
#pragma bindingSpecification pushbutton,listbox,"Delete all rows in %2 when %1 is pushed"
BindSource=pushbutton(source)
bindTarget=listbox(target)
BindSource.addActionNotificationReceiver self
```

   Die #pragma-Anweisung legt fest, was in der Objektverknüpfung-Dialogbox erscheinen soll, wenn der Anwender die beiden Objekte verknüpft, indem er die Maus bei gedrückter Befehls-, Shift- und Maustaste vom Quell- zum Zielobjekt (also vom PushButton zur ListBox) zieht. Die #pragma-Anweisung bestimmt die Quell- und die Zielklasse und enthält den Text, der im Objektverknüpfung-Dialog erscheint. %1 wird durch den Namen des Quell-Steuerelements, %2 durch den Namen des Ziel-Steuerelements ersetzt.

   Wenn die Verknüpfung zutrifft (also der PushButton gedrückt wird), werden die verknüpften Steuerelemente an die Bind-Methode übergeben, und zwar als generische Objekte. Um die Objekte in den BindSource- und BindTarget-Eigenschaften zu speichern, die Sie in Schritt 5 angelegt haben, müssen sie die Objekte mittels Typkonvertierung wieder zu einem PushButton und einer ListBox machen. Die Programmzeilen 2 und 3 der Bind-Methode erledigen genau dies.

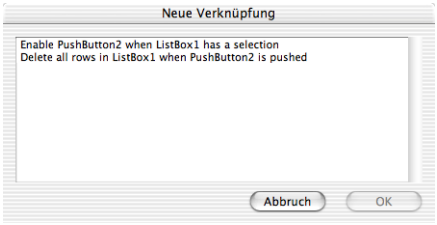
   Die letzte Programmzeile stellt die Verbindung zwischen der DeleteAllRowsBind-Klasse und dem Action-Event des PushButtons her, indem die Methode addActionNotificationReceiver der PushButton-Klasse aufgerufen wird. Als Parameter wird „self“ übergeben, wodurch die DeleteAllRowsBind-Klasse selbst repräsentiert wird.

   Jetzt fehlt noch der Programmcode, der die Zeilen der ListBox löscht, wenn der Anwender den PushButton anklickt:
7. Fügen Sie dazu die neue Methode „PerformAction“ zu Ihrer Klasse hinzu.
8. Der Programmcode dieser PerformAction-Methode sieht so aus:

```
bindTarget.DeleteAllRows
```

   Um die neue Objektverknüpfung auszuprobieren, platzieren Sie eine ListBox und einen PushButton auf einem Fenster und füllen die ListBox mit einigen Zeilen. Verknüpfen Sie die beiden Steuerelemente, indem Sie bei gedrückter Shift-, Befehls- und Maustaste die Maus vom PushButton zur ListBox ziehen. Wenn der Dialog „Neue Verknüpfung“ erscheint, wird auch die selbsterzeugte Verknüpfung aufgelistet (siehe Abbildung 238).

Abb. 238: Die selbstdefinierte Verknüpfung in der Liste der Verknüpfungen



9. Wählen Sie den Eintrag **Delete all rows from listbox1 when pushbutton1 is pushed** und testen Sie die neue Verknüpfung in der Laufzeitumgebung.

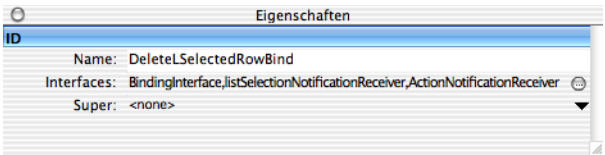
Die Verknüpfung „Lösche die selektierte Zeile"

Um eine Verknüpfung zu erzeugen, die dafür sorgen soll, dass die selektierte Zeile einer ListBox gelöscht wird, sobald ein PushButton gedrückt wird, legen Sie in Ihrem Projekt eine neue Klasse an und wiederholen Sie alle oben beschriebenen Schritte.

Da die neue Version der Verknüpfung nur aktiv werden soll, wenn eine Zeile selektiert ist, müssen einige zusätzliche Interfaces unterstützt werden, damit man mitbekommt, ob der Anwender eine Zeile auswählt.

1. Legen Sie in Ihrem Projekt die Klasse DeleteSelectedRowBind an.
2. Ergänzen Sie listSelectionNotificationReceiver bei der Interfaces-Eigenschaft der Klasse. Das Eigenschaftenfenster sollte dann aussehen wie in Abbildung 239.

Abb. 239: Das Eigenschaftenfenster der Klasse DeleteSelectedRowBind



3. Versehen Sie die Klasse mit der neuen Methode „Bind". Geben Sie der Methode die Parameter „Source as Object, Target as Object".
4. Geben Sie den folgenden Programmcode ein:

```
#pragma bindingSpecification pushbutton,listbox,"Delete the selected row in %2 when %1 is pushed"
BindSource=pushbutton(source)
bindTarget=listbox(target)

BindTarget.addListSelectionNotificationReceiver self
BindSource.addActionNotificationReceiver self
BindSource.enabled=(bindTarget.listindex >= 0)
```

Die erste neue Programmzeile sorgt dafür, dass die DeleteSelectedRowBind-Klasse über jede Veränderung der Selektion in der ListBox informiert wird. Die letzte Zeile aktiviert das Quellobjekt (den PushButton), wenn beim Öffnen des Fensters eine Zeile des Zielobjektes (der ListBox) selektiert ist und deaktiviert das Quellobjekt, wenn im Zielobjekt keine Selektion vorliegt.

Nun muss der Programmcode hinzugefügt werden, der den PushButton aktiviert bzw. deaktiviert, wenn der Anwender im geöffneten Fenster Einträge der ListBox selektiert bzw. deselektiert. Dazu muss die Klasse um die Methoden erweitert werden, die Teil des ListSelectionNotificationReceiver-Interfaces sind.

5. Erweitern Sie die Klasse um die Methoden „SelectionChanged“ und „SelectionChanging“. Beide haben keine Parameter. Schreiben Sie folgenden Programmcode in die SelectionChanged-Methode:
```
bindSource.enabled=(bindTarget.listindex >= 0)
```
Immer, wenn sich die Selektion in der ListBox verändert, wird die Verknüpfung aktiv und die SelectionChanged-Methode aufgerufen.

Die SelectionChanging-Methode hat keinen Programmcode.

Jetzt fehlt nur noch der Programmcode, der dafür sorgt, dass die ausgewählte Zeile gelöscht wird, wenn der Anwender auf den PushButton klickt.

6. Fügen Sie die PerformAction-Methode hinzu und geben Sie folgende Programmzeile ein:
```
bindTarget.removeRow bindTarget.listindex
```
Schließlich können Sie einen weiteren Knopf auf dem Fenster ablegen und ihn mit der ListBox verknüpfen.

7. Wählen Sie die neue Verknüpfung aus und testen Sie sie in der Laufzeitumgebung.

Exportieren und Importieren von Klassen

Klassen können exportiert und importiert werden. Wird eine Klasse exportiert, erscheint sie auf dem Desktop mit einem Würfelsymbol wie in Abbildung 240. Um eine Klasse zu importieren, ziehen Sie einfach das Symbol auf Ihr Projektfenster. Dabei wird eine Kopie der Klasse in Ihrem Projekt angelegt. Das Projekt ist also nicht mit der Datei auf dem Desktop verknüpft. Sie können diese wegwerfen, wenn Sie sie nicht für ein anderes Projekt benötigen.

Abb. 240: Eine exportierte Klassen-Datei

Ist eine importierte Klasse von einer anderen Klasse abhängig, so muss diese andere Klasse ebenfalls in dem Projekt vorhanden sein. Dies ist natürlich nicht von Bedeutung, wenn die zugrunde liegende Klasse zu den in REALbasic eingebauten Klassen gehört (wie z.B. EditField etc.).

Exportierte Klassen können verschlüsselt werden, so dass man sie zwar importieren, nicht aber den Quelltext ansehen oder verändern kann. Wenn die Klasse verschlüsselt ist, wird das Icon der Klasse im Projektfenster mit einem kleinen Schlüssel-Symbol dargestellt.

Abb. 241: Eine verschlüsselte Klasse

Weitere Informationen entnehmen Sie dem Abschnitt „Verschlüsseln des Quelltextes“ auf Seite 332.

Importieren externer Projektelemente

Wenn Sie eine Klasse in mehr als einem Projekt verwenden möchten, dann können Sie diese Ihrem Projekt als externes Projektelement hinzufügen. Ein externes Projektelement wird auf der Festplatte gespeichert und jedes Projekt, das davon Gebrauch macht, führt einen Verweis darauf. Wenn die Klasse in einem Projekt eine Änderung erfährt, dann

schlägt sich diese Veränderung auch in allen anderen Projekten, die auf das externe Element verweisen, nieder. Veränderungen, die Sie an einem externen Projektelement vornehmen, werden mit dem Speichern des Projekts gespeichert.

Um Ihrem Projekt ein externes Element hinzuzufügen, halten Sie die Befehls (⌘)- und Wahltaste gedrückt (Windows: Strg+Shift), während Sie das Element vom Schreibtisch auf das Projektfenster ziehen. Im Projektfenster wird der Name des externen Elements in Kursivschrift angezeigt.

Detaillierte Informationen zu diesem Thema finden Sie im Abschnitt „Externe Projektelemente“ auf Seite 83.

Ob die Klasse geschützt ist, können Sie nach dem Import feststellen, indem Sie einen Doppelklick auf die Klasse im Projektfenster ausführen. Sollte die Klasse geschützt sein, weist eine Dialogbox Sie darauf hin.

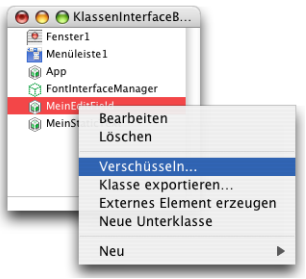
Es gibt zwei Arten des Exports. Sie können die Klasse einfach aus dem Projektfenster auf den Desktop ziehen. Dabei entsteht eine Datei mit dem Namen der Klasse wie in Abbildung 240. Wenn man den Ordner, in den exportiert werden soll, auf dem Desktop nicht sehen kann, dann kann der Export auch über **Ablage/Klasse exportieren** erfolgen.

Verschlüsseln des Quelltextes

Wenn Sie anderen REALbasic-Benutzern eine selbstentwickelte Klasse, aber nicht deren Quelltext zur Verfügung stellen wollen, können Sie vor dem Export den Quelltext der Klasse verschlüsseln. Das erzeugt beim Export eine Klasse, die man importieren und benutzen, jedoch nicht anschauen oder bearbeiten kann. Das ist vor allem dann interessant, wenn Sie ausgefeilte Klassen entwickeln, die Sie als Add-Ons verkaufen wollen.

Zum Verschlüsseln der Klasse klicken Sie im Projektfenster auf dieselbe und rufen **Bearbeiten/Verschlüsseln** auf. Alternativ können Sie auch einen ctrl-Klick (Windows: Rechtsklick) auf den Namen der Klasse machen und im Kontextmenü „Verschlüsseln“ aufrufen.

Abb. 242: Das Kontextmenü im Projektfenster



Es erscheint folgende Dialogbox:

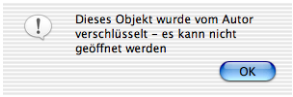
Abb. 243: Der Verschlüsseln-Dialog

Geben Sie ein Passwort für die Verschlüsselung ein und bestätigen Sie dieses. Selektieren Sie **V3 Verschlüsselung oder höher verw.** nur, wenn die Klasse mit der Version 3 oder höher von REALbasic verwendet wird. Deselektieren Sie die Option nur, wenn Sie die Klasse mit REALbasic 3.0 oder kleiner verwenden wollen.

Wichtig: Vergessen Sie ihr Passwort nicht!

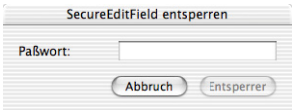
Das Symbol einer verschlüsselten Klasse im Projektfenster wird mit einem Schlüssel versehen. Wenn ein Anwender versucht, eine verschlüsselte Klasse im Fenstereditor zu öffnen, erscheint folgende Warnbox:

Abb. 244: Der Dialog „Objekt verschlüsselt"



Um die Klasse oder ihren Code zu bearbeiten, müssen Sie sie mit dem Passwort entschlüsseln. Selektieren Sie dazu die Klasse im Projektfenster und rufen Sie den Menüpunkt **Bearbeiten/Entschlüsseln** auf. Es erscheint der Entschlüsseln-Dialog.

Abb. 245: Der Entschlüsseln-Dialog



Geben Sie das Passwort ein und klicken Sie auf **Entschlüsseln**. Nach kurzer Zeit verschwindet der Schlüssel aus dem Symbol der Klasse, um anzuzeigen, dass die Entschlüsselung erfolgreich war. Wenn Sie das Passwort nicht kennen oder vergessen haben, gibt es keine Möglichkeit, die Klasse zu entschlüsseln.

Wenn Sie eine verschlüsselte Klasse weitergeben oder verkaufen wollen, müssen Sie eine Liste ihrer Methoden und Eigenschaften zur Verfügung stellen, da es ja ohne Passwort nicht möglich ist, den Quelltext einzusehen.

Löschen einer Klasse

Bevor Sie eine Klasse löschen, sollten Sie ganz sicher sein, dass diese weder von anderen Klassen benutzt wird, noch als Oberklasse für andere Klassen dient.

So entfernen Sie eine Klasse aus einem Projekt:

1. Wählen Sie die Klasse im Projektfenster aus.
2. Drücken Sie entweder die Backspace-Taste, rufen Sie den Menüpunkt **Bearbeiten/Löschen** auf oder führen Sie einen ctrl-Klick (Windows: Rechtsklick) auf die Klasse aus und rufen im Kontextmenü **Löschen** auf.

Diese Aktion lässt sich mit **Bearbeiten/Undo** (⌘-Z bzw. Strg-Z) rückgängig machen.

Kapitel 10 Datenbankprogrammierung mit REALbasic

Mit REALbasic können Sie Datenbankanwendungen programmieren, die auf verschiedene Datenbank-Engines aufsetzen können, z.B. auch auf die mitgelieferte Datenbank-Engine von REAL Software.

Die Datenbank von REAL Software, genannt REALdatabase, in der Standard- und Professional-Version von REALbasic enthalten und wird vollständig unterstützt.

Die Verbindung mit Datenbank-Servern wird nur von REALbasic Professional unterstützt.

Inhalt

- Die Datenbankarchitektur von REALbasic
- Die Datenbankwerkzeuge in REALbasic
- Anlegen und Modifizieren von Datenbanken im Projektfenster
- Das DataControl Steuerelement
- Verwenden von Objektverknüpfungen
- Programmieren eines Datenbank-Frontends

Die Datenbankarchitektur von REALbasic

Sie können mit REALbasic ein Frontend zu Ihrer Datenbank programmieren. Dieses Frontend arbeitet mit einem Datenbank-Backend (einer Datenbank-Engine) zusammen, das für die eigentlichen Datenbankoperationen, wie das Speichern der Daten, zuständig ist. Die Rolle des Datenbank-Backends kann von einer separaten Applikation oder von der in REALbasic integrierten Datenbank-Engine übernommen werden.

Das Frontend dient dabei als Benutzerschnittstelle und Hilfsmittel, mit dem Abfragen an die Quelle gestellt und Informationen angezeigt und gedruckt werden. Der Endanwender verwendet das Frontend, um Datensätze zu betrachten, einzugeben, zu modifizieren, zu suchen, zu sortieren und auszudrucken.

Sie können ein zusätzliches Datenbankprogramm, wie z.B. 4D Server von 4th Dimension oder Oracle einsetzen, um die Daten zu verwalten. Die Datenbankapplikation, die für die tatsächliche Datenverwaltung zuständig ist, wird als „Datenquelle“ bezeichnet.

Der große Vorteil dieser Architektur besteht darin, dass ein mit REALbasic programmiertes Datenbank-Frontend mit allen unterstützten Datenquellen zusammenarbeiten kann. Sie können zur Entwicklung einer Datenbankanwendung die interne REAL-Datenbank-Engine verwenden und später das System an höhere Anforderungen (z.B. Multi-User-Betrieb) anpassen, indem Sie die Datenquelle wechseln.

Ein REALbasic Datenbank-Frontend kann auch mit mehreren verschiedenen Datenquellen gleichzeitig zusammenarbeiten. Beispielsweise könnten lokale Daten von 4D Server geliefert werden, während über eine Remote-Verbindung Daten eines SQL-Servers oder einer ODBC-Datenbank bezogen werden.

Zur Unterstützung der verschiedenen Datenquellen gibt es spezielle Datenbank-Plugins von REAL Software und von Drittanbietern. Die Datenbank-Plugins müssen sich im Plugins-Verzeichnis innerhalb des REALbasic-Verzeichnisses befinden, damit Sie sie in Ihrem Projekt verwenden können. Eine Ausnahme bildet die REALdatabase, die integraler Bestandteil von REALbasic ist.

Es ist möglich, zusätzliche Datenquellen zu unterstützen, indem entsprechende Plugins für die jeweiligen Backends entwickelt werden. Das Plugin-SDK von REAL Software liefert die notwendigen Informationen, um eigene Datenbank-Plugins zu schreiben.

Strukturierte Abfragesprache

Ein REALbasic Datenbank-Frontend verwendet zur Kommunikation mit seiner Datenquelle die Strukturierte Abfragesprache SQL (Structured Query Language). Das Datenbank-Plugin, das Sie zum Zugriff auf Ihre Datenquelle verwenden, empfängt eine SQL-Anweisung von REALbasic, übersetzt diese – falls notwendig – in eine für Ihre Datenquelle verständliche Form und sendet diese an die Datenquelle.

Die mitgelieferte REALdatabase verwendet die in diesem Abschnitt beschriebene Untermenge des SQL-Standards. Im Fall der REALdatabase wird das Plugin als integraler Bestandteil von REALbasic mitgeliefert, muss also nicht als zusätzliche Datei im Plugins-Verzeichnis installiert werden.

Fast alle von REALbasic unterstützten Datenbankserver verfügen über native SQL-Backends (die einzige Ausnahme ist 4DServer). Wenn Sie mit einem nativen SQL-Backend arbeiten, übergibt das REALbasic-Plugin Ihren SQL-Befehl unmodifiziert an die Datenquelle. Das heißt, dass Sie alle für diese Datenquelle gültigen SQL-Befehle in Ihrem REALbasic-Frontend verwenden können. Welche SQL-Kommandos und Datentypen Ihr Datenbankserver unterstützt, entnehmen Sie bitte dessen Dokumentation.

Die von der REALDatabase und dem 4th Dimension-Plugin unterstützte SQL-Untermenge wird in Anhang A der Sprachreferenz (gedruckte und PDF-Version) dokumentiert.

Hinweis: Wenn Sie ein Front-End für einen kommerzielles Back-End schreiben, steht Ihnen wahrscheinlich ein erweiterter SQL-Sprachumfang zur Verfügung. Auch wenn Ihr Front-End-Prototyp die REALDatabase verwendet, sollte es ohne Änderungen mit einer anderen Datenquelle funktionieren. Nach dem Umstellen der Datenquelle können Sie dann den zusätzlichen Sprachumfang nutzen und Ihr Front-End an die erweiterten Features anpassen.

Falls Sie sich bisher noch nicht mit SQL beschäftigt haben, sollten Sie zunächst einige SQL-Grundlagen erlernen, bevor Sie mit der Implementierung Ihrer REALbasic-Datenbankanwendung beginnen. Dieses Handbuch unternimmt nicht den Versuch, Ihnen SQL beizubringen. Bitte konsultieren Sie statt dessen eines der vielen erhältlichen SQL-Referenzwerke, z.B. *SQL für Dummies* von Allen G. Taylor (ISBN: 3-8266-2787-3) oder das SQL-Tutorial, das Sie online unter folgender URL finden: http://w3.one.net/~jhoffman/sqltut.htm.

Datenbank-Werkzeuge in REALbasic

Ein Datenbank-Frontend verwendet typischerweise eine Kombination aus datenbankspezifischen und generischen Steuerelementen und Befehlen.

Mit dem DatabaseQuery und dem DataControl gibt es zwei datenbankspezifische Steuerelemente. Außerdem stehen mehrere datenbankspezifische Klassen und Methoden zur Verfügung.

Die REALDatabase kann die Tabellen einer Datenbank entweder in einem virtuellen Volume (siehe „Virtuelle Volumes“ auf Seite 299) oder als einzelne Dateien speichern.

In einer REALdatabase können Sie Tabellen hinzufügen, löschen oder modifizieren (z.B. Felder und Indizes anlegen oder löschen), ohne dass davon nicht betroffene Daten verloren gehen oder umkopiert werden müssen.

Die REALDatabase-Datenbank Engine unterstützt verschiedene Textcodierungen. Wenn Sie Datensätze einfügen, speichert die Datenbank die Codierung der Textfelder und verwendet diese auch beim Auslesen der Felder automatisch. Es ist also im Gegensatz zu den meisten anderen Datenbanken nicht nötig, beim Auslesen von Daten aus einer REALDatabase die Textcodierung anzugeben.

REALDatabase unterstützt Transaktionen für Änderungen der Daten und der Datenbankstruktur. Eine Transaktion wird automatisch initiiert, wenn Sie Änderungen an der Datenbank oder den Daten vornehmen und wird durch einen Aufruf von COMMIT oder ROLLBACK abgeschlossen.

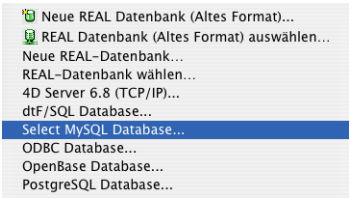
Jede Tabelle in REALDatabase besitzt eine spezielle Spalte namens „_rowid", die einen eindeutigen numerischen Schlüssel für jeden Datensatz mitführt. Dieser wird automatisch gesetzt und kann auch für Verknüpfungen (Joins) verwendet werden.

Keine Tabelle darf größer als 2 GB werden. Wenn Sie die Datenbank auf einem virtuellen Volume speichern, darf die Gesamtgröße der Datenbank 2 GB nicht überschreiten. Da „_rowid" mit jedem neuen Datensatz automatisch um eins vergrößert wird und ein einmal vergebener Wert nicht erneut benutzt wird, können Sie maximal ca. 4 Milliarden Datensätze (2^{32}) in eine Tabelle einfügen.

Auswahl einer Datenquelle

Sie können eine Datenquelle entweder mittels Programmcode auswählen oder über den Menüpunkt **Ablage/Datenquelle hinzufügen** zu Ihrem Projekt hinzufügen. Unter diesem Menüpunkt ist standardmäßig nur die REALdatabase aufgeführt. Andere Datenbanken erscheinen dort nur, wenn Sie ein entsprechendes Datenbank-Plugin im Plugins-Ordner innerhalb des REALbasic-Ordners installiert haben.

Abb. 246: Das Untermenü Datenquelle hinzufügen

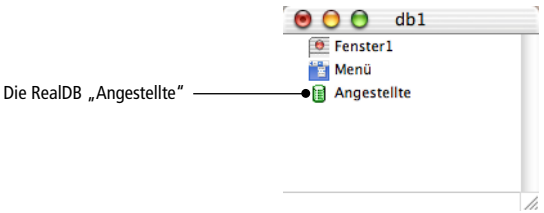


Mit **REAL-Datenbank wählen…** öffnet sich ein Dialog, in dem Sie eine vorhandene Datenbank auswählen können.

Der Menüpunkt **Neue REAL-Datenbank…** zeigt einen Sichern-Dialog zum Anlegen einer neuen Datenbank.

Die ausgewählte oder neu angelegte Datenquelle erscheint im Projektfenster, wie in Abbildung 247 zu sehen.

Abb. 247: Eine Datenbank im Projektfenster



Hinweis: Sie können Datenbanken auch einfach in das Projektfenster ziehen, um sie zum Projekt hinzuzufügen.

Sie können eine Datenbank aus dem Projektfenster entfernen, indem Sie sie auswählen und den Menüpunkt **Bearbeiten/Löschen** aufrufen.

Selbstverständlich können Sie eine REALdatabase auch mittels Programmcode öffnen. Erzeugen Sie dazu eine Instanz der REALdatabase-Klasse und weisen deren DatabaseFile-Eigenschaft den Pfad der Datenbankdatei zu. Rufen Sie dann die Methode „Connect" auf. Kehrt „Connect" mit True zurück, ist eine Verbindung mit der Datenbank hergestellt und Sie können Datenbankoperationen ausführen. Liefert „Connect" hingegen False zurück, konnte keine Verbindung hergestellt werden. Dann finden Sie in den Eigenschaften „ErrorCode" und „ErrorMessage" Hinweise auf die Fehlerursache.

Hier ein Beispiel:

```
Dim dbFile as FolderItem
Dim db as REALDatabase
db=New REALDatabase
dbFile = GetFolderItem("Pubs")
db.DatabaseFile=dbFile
If db.Connect() then
  // Datenbankoperationen bei erfolgreichem Verbindungsaufbau
else
  Beep
  MsgBox "Fehlermeldung: "+db.ErrorMessage
end if
```

Im Beispiel wird eine Datenbank namens „Pubs“ geöffnet, die sich im selben Verzeichnis wie REALbasic befindet. Sollte sich die Datenbank in einem anderen Verzeichnis befinden, können Sie die Volumes-Funktion und die Parent- und Child-Eigenschaften der FolderItem-Klasse verwenden, um den Zugriffspfad festzulegen. Beispiele dazu finden Sie im Abschnitt „Lesen einer Datei an einer vorgegebenen Stelle“ auf Seite 284.

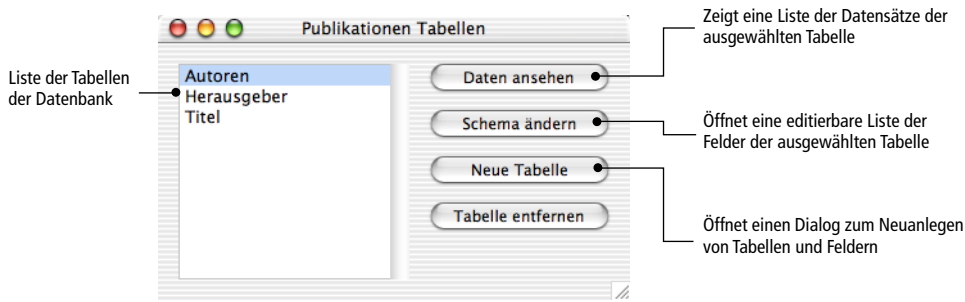
Zum Erzeugen einer neuen Datenbank gehen Sie fast genauso vor. Der Unterschied besteht darin, dass Sie statt Connect die Methode CreateDatabaseFile aufrufen müssen. Wird True zurückgegeben, wurde die Datenbank erfolgreich erzeugt. Hier ist ein Beispiel:

```
Dim db as REALDatabase
Dim f as FolderItem
f=New FolderItem("meineDatenbank")
db=New REALDatabase
db.databaseFile=f
If db.CreateDatabaseFile then
  //proceed with database operations...
else
  MsgBox "Error message: "+db.ErrorMessage
end if
```

Datenbanken im Projektfenster anlegen und modifizieren

Mittels Doppelklick auf eine Datenbank im Projektfenster können Sie das Schema (die Struktur) der Datenbank öffnen. Es wird eine Liste aller Tabellen der Datenbank angezeigt. Ausgehend von dieser Liste können Sie die eingegebenen Daten und die Liste der Felder (Spalten) und Feldattribute betrachten.

Abb. 248: Das Schema einer Datenbank



Bei einer neuen Datenbank ist die Tabellenliste zunächst leer. Eine neue Tabelle wird folgendermaßen angelegt:

1. Klicken Sie auf den Knopf **Neue Tabelle** in der Dialogbox „Neue Tabelle“.

Abb. 249: Der Dialog „Neue Tabelle“

2. Geben Sie den Tabellennamen ein und drücken Sie Tab.

   Sie müssen in der Tabelle mindestens ein Feld anlegen. Das Feld „_rowid“ wird automatisch erzeugt, Sie brauchen es nicht explizit anzulegen und können an Stelle eines Primärschlüssels verwenden.
3. Unter Feldname geben Sie den Namen des ersten Feldes ein und drücken Tab. Jetzt ist der Knopf „Anlegen“ aktiv.
4. Wählen Sie aus dem Popup einen Datentyp aus. Die REAL-Datenbank unterstützt folgende Datentypen: Integer, Varchar, Double, Date, Time, Timestamp, Boolean, String und Binary (Größe bis zu 2^{31} Bytes). Andere Datenquellen unterstützen evtl. weitere Datentypen. Beim Versuch, einen von der konkreten Datenquelle nicht unterstützten Datentyp zu verwenden, erscheint die Meldung **Unbekannter Datentyp**.
5. Klicken Sie auf **Anlegen**, um das erste Feld zu erzeugen. Wenn Sie dann auf **OK** klicken, erscheint das Feld in der Feldliste, wie in folgender Abbildung zu sehen.

Abb. 250: Die Dialogbox „Neue Tabelle“ mit einem Feld

Die REALdatabase unterstützt keine Feldattribute wie „Mussfeld“ und „Primärschlüssel“. Werden diese von einem anderen Back-End unterstützt, können Sie sie über den Eigenschaften-Knopf aktivieren.

6. Fügen Sie der Tabelle durch Wiederholen der Schritte 3 bis 5 weitere Felder hinzu.

Indizes hinzufügen

Nachdem Sie die Felder hinzugefügt haben, können Sie für ausgewählte Felder einen Index anlegen. Indizes beschleunigen Datenbankzugriffe.

Beachten Sie: Bestimmte Datenquellen unterstützen keine Indizierung. Für solche Datenquellen wird im **Neue Tabelle/Tabelle bearbeiten**-Dialogfeld kein „Indizes“-Knopf angezeigt.

Sie sollten in jeder Tabelle einen Primärschlüssel und die Felder indizieren, über die Sie die meisten Such- und Sortiervorgänge ausführen. Die Abfrage von Informationen und relationale Operationen zwischen Tabellen gehen schneller vonstatten, wenn die verwendeten Felder indiziert sind.

Felder, die nur wenige Werte annehmen können (z.B. Geschlecht oder Anrede), Felder, in denen selten gesucht wird und Felder in Tabellen mit wenigen Datensätzen müssen nicht indiziert werden, da hier die Such- und Sortierzeit nicht ins Gewicht fällt.

Sie sollten nicht zu viele Felder indizieren, da jeder Index die Datenbank vergrößert und die Datenbank zum Aktualisieren der Indizes Zeit benötigt, wenn Datensätze hinzugefügt oder gelöscht werden.

Um Indizes für eine Tabelle zu erzeugen, gehen Sie folgendermaßen vor:

1. Klicken Sie auf den Knopf **Indizes** in der Tabellen-Dialogbox.

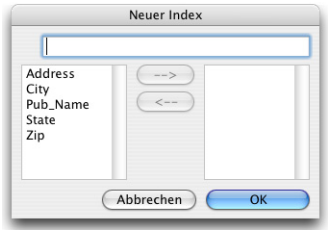
   Hinweis: Falls die Datenquelle keine Indizierung unterstützt, erscheint an dieser Stelle kein Indizes-Knopf.

2. Es erscheint die „Indizes bearbeiten"-Dialogbox.

Abb. 251: Die Indizes-Dialogbox

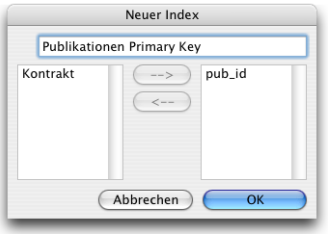
3. Klicken Sie auf **Neu**..., um die Dialogbox „Neuer Index" aufzurufen.

Abb. 252: Die Dialogbox „Neuer Index"



4. Selektieren Sie das erste zu indizierende Feld und klicken Sie auf den Rechtspfeil oder führen Sie einen Doppelklick auf das Feld aus, um es in die rechte Spalte zu übernehmen.
5. *Optional:* Um einen zusammengesetzten Index (einen Index über zwei oder mehr Felder) zu erzeugen, übernehmen Sie weitere Felder in die rechte Liste. Z.B. könnten Sie in einer Adresstabelle einen zusammengesetzten Index aus den Feldern Nachname und Vorname erzeugen, damit die Datenbank schnell zwischen verschiedenen Personen mit dem gleichen Nachnamen unterscheiden kann.
6. Geben Sie dem Index im oberen Eingabefeld einen Namen. Folgende Abbildung zeigt einen „fertigen" Index.

Abb. 253: Der Index für das Publikationen-Primärschlüsselfeld



7. Klicken Sie auf **OK**, um den Index in die Datenbank zu übernehmen.

   Es erscheint wieder die Dialogbox „Indizes bearbeiten“ und zeigt den Namen des neu erzeugten Index.
8. *Optional:* Klicken Sie auf **Neu...**, um weitere Indizes zu erzeugen und wiederholen Sie die Schritte 4 bis 7.

Datensätze anzeigen

Der Knopf „Daten ansehen“ im Datenbankschema-Dialog (Abbildung 248 auf Seite 337) zeigt eine Liste mit den Datensätzen der gewählten Tabelle. Der Dialog akzeptiert eine SQL-Abfrage (SQL-Query), die Sie an die Datenbank richten können. Die standardmäßig vorgegebene SQL-Abfrage zeigt alle Datensätze der Tabelle an. Sollen nur bestimmte Datensätze angezeigt werden, können Sie dazu eine WHERE-Bedingung formulieren. Sollen nur bestimmte Felder angezeigt werden, können Sie diese an Stelle des „*“ auflisten. Mit ORDER BY können Sie darüber hinaus eine Sortierung nach einem oder mehreren Feldern festlegen.

Datentypen für Datenbankfelder

Unten sehen Sie eine Aufstellung der von der REALdatabase unterstützten Datentypen.

Hinweis: Welche Datentypen Sie in einer konkreten Datenbank verwenden können, ist von der jeweiligen Datenbank-Engine abhängig. Native SQL-Datenbanken können weitere Datentypen unterstützen. Welche das sind, können Sie der Dokumentation Ihres Datenbankservers entnehmen.

Tabelle 36. Von der REALdatabase unterstützte Datentypen

Feldtyp	Numerischer Code	Beschreibung
Integer	3	Numerischer Ganzzahl-Datentyp. Die maximale Stellenanzahl ist implementierungsabhängig. Die REALdatabase verwendet 4-Byte-Integer-Werte (Wertebereich ±2,000,000.000). Für andere Datenquellen konsultieren Sie bitte deren Dokumentation.
Text oder VarChar	5	Speichert Textdaten, bei denen die Anzahl der Zeichen von Datensatz zu Datensatz variiert, ungenutzte Zeichen aber nicht mit Leerzeichen überschrieben werden. VARCHAR(20) legt also ein VARCHAR-Feld mit der Maximallänge von 20 Zeichen an. Die REAL-Database unterstützt den VarChar-Feldtyp, nicht den Text-Typ. Die theoretische Maximallänge eines VarChar-Feldes beträgt 2^{31} Bytes. Wenn Sie in einem CREATE TABLE oder ALTER TABLE-Befehl eine Maximallänge festlegen, wird diese ignoriert.
Double	7	Speichert Fließkommazahlen mit doppelter Genauigkeit. Die REAL-Database verwendet 8-Byte Doubles.
Date	8	Speichert Jahr, Monat und Tag eines Datums im Format JJJJ-MM-TT.
Time	9	Speichert die Stunden, Minuten und Sekunden einer Uhrzeit im Format HH:MM:SS. Die Stunden und Minuten sind zweistellig. Die Sekunden sind auch zweistellig, können aber einen optionalen Nachkommateil haben, z.B. 09:55:25.248. Die Standardlänge des Nachkommateils ist 0.

Tabelle 36. Von der REALdatabase unterstützte Datentypen

Feldtyp	Numerischer Code	Beschreibung
TimeStamp	10	Speichert Datum und Uhrzeit im Format JJJJ-MM-TT HH:MM:SS. Die Einzelkomponenten entsprechen den Datentypen Date und Time mit dem Unterschied, dass der Nachkommateil der Sekunden standardmäßig 6 und nicht 0 Stellen lang ist. Wenn ein TimeStamp-Wert keinen Nachkommaanteil hat, ist er 19 Stellen lang. Verfügt er über einen Nachkommaanteil, ist seine Länge 20 Stellen plus die Anzahl der Nachkommastellen.
Boolean	12	Speichert die Werte von TRUE und FALSE.
Binary	14	Speichert Code, Bilder und hexadezimale Daten. Schlagen Sie in der Dokumentation Ihrer Datenquelle nach, um die maximale Länge des Binary-Feldes zu erfahren. Die REALdatabase speichert bis zu 2^{31} Bytes.
String	18	Bis zu 2^{31} Bytes Text. Identisch mit VarChar.

Die REALdatabase unterstützt ein Fluchtsymbol („Escape Character“) in Zeichenketten. Ein Fluchtsymbol entwertet ein direkt darauf folgendes Sonderzeichen und sorgt dafür, dass dieses als normales Zeichen gewertet wird. Das Fluchtsymbol kann mittels SET ESCAPE frei definiert werden. Z.B. stellt „SET ESCAPE p“ den Backslash als Fluchtsymbol ein. Mit „SET ESCAPE“ ohne Argument wird das Fluchtsymbol wieder deaktiviert.

Das DatabaseQuery Steuerelement

Das DatabaseQuery Steuerelement kann verwendet werden, um Abfragen an die Datenquelle zu senden. Alternativ können Sie diese Funktion auch „von Hand“ programmieren.

Abb. 254: Das DatabaseQuery Steuerelement

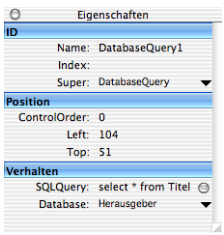
Ein DatabaseQuery kann wie jedes andere Steuerelement aus der Steuerelementepalette auf ein Fenster gezogen werden. Im Unterschied zu den meisten anderen Steuerelementen ist es allerdings für den Anwender nicht sichtbar.

Tabelle 37. Eigenschaften von DatabaseQuery

Name	Beschreibung
Database	Die Datenquelle, an die eine Abfrage gestellt wird.
SQLQuery	Der Text der SQL-Abfrage, die über *Database* ausgeführt werden soll.

SQLQuery wird automatisch ausgeführt, wenn das Fenster dargestellt wird. Die in Abbildung 255 gezeigten Eigenschaften fordern beispielsweise alle Zeilen und Spalten der Titel-Tabelle an, sobald das Fenster geöffnet wird. Allerdings ist das DatabaseQuery nicht in der Lage, die Zeilen und Spalten ohne fremde Hilfe darzustellen.

Abbildung 255. Die Eigenschaften eines DatabaseQuery Steuerelements:



DatabaseQuery verfügt über die Methode RunQuery, die *SQLQuery* auf *Database* anwendet.

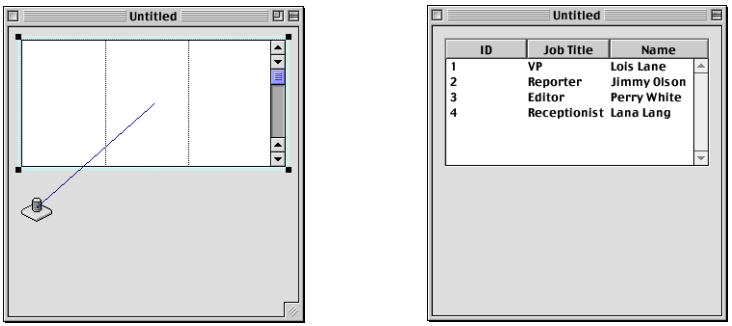
DatabaseQuery und Objektverknüpfungen

Mit dem DatabaseQuery-Steuerelement und Objektverknüpfungen können Sie ein einfaches Datenbank-Frontend zusammenbauen, ohne eine einzige Zeile programmieren zu müssen.

Da *SQLQuery* automatisch vom DatabaseQuery-Steuerelement ausgeführt wird, müssen Sie sich nur noch um das Anzeigen des Ergebnisses kümmern. Erzeugen Sie zu diesem Zweck eine ListBox im Fenster und verknüpfen Sie DatabaseQuery- und ListBox-Steuerelement miteinander. Halten Sie dazu die Shift- und Befehlstaste (Windows: Shift und Strg) gedrückt und ziehen Sie eine Verbindungslinie zwischen den Steuerelementen. Der Dialog „Neue Objektverknüpfung“ erscheint. Wählen Sie die Aktion `Verbinden Sie ListBox1 mit den Listendaten von DatabaseQuery1...`.

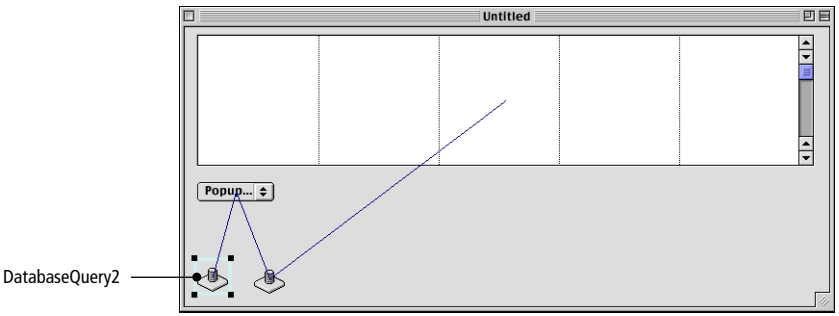
Bei Aufruf von **Debug/Programm starten** erscheinen die Datensätze in der ListBox, wie in Abbildung 256 gezeigt.[1]

Abb. 256: Eine einfache Datenbank ohne Programmierung



Genauso können Sie ein DatabaseQuery-Steuerelement mit einem Popup-Menü verknüpfen. Das Popup-Menü wird dann mit den Ergebnissen der Abfrage gefüllt. Sie können anders herum auch das Popup mit dem DatabaseQuery-Steuerelement verknüpfen, so dass Sie im Popup durch Auswahl eines anderen Eintrags die Abfrage modifizieren können. Abbildung 257 illustriert dieses Konzept:

Abb. 257: Verwenden einer Objektverknüpfung, um die Einträge eines PopupMenu-Steuerelements zu belegen

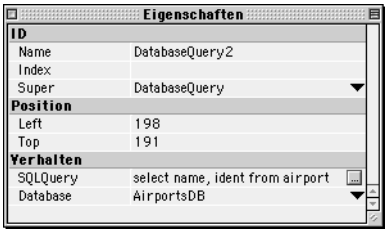


Das ausgewählte DatabaseQuery Steuerelement enthält folgende SQL-Abfrage im Eigenschaftenfenster:

```
Select name, ident from airports
```

1. Die Spaltenüberschriften wurden im Eigenschaftenfenster der ListBox eingegeben.

Abb. 258: Die SQL-Abfrage, die das PopupMenu mit Daten belegt



Die SQL-Abfrage extrahiert einen Namen und eine ID aus der Datenbank. Die Verknüpfung trägt den Namen in das PopupMenu ein und weist der unsichtbaren RowTag-Eigenschaft die ID zu. Die RowTag-Eigenschaft ist eine unsichtbare Spalte, die als Erkennungsmerkmal (Identifier) benutzt werden kann.

Die Verknüpfung mit dem PopupMenu-Steuerelement lautet:

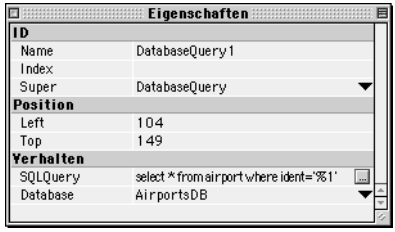
```
Bind Popup Menu1 with list data from DatabaseQuery's results
```

Die Verknüpfung des PopupMenu-Steuerelements mit dem anderen DatabaseQuery lautet:

```
Bind DatabaseQuery1 parameter with string data from PopupMenu1 RowTag
```

Diese übergibt den Wert der RowTag-Eigenschaft des PopupMenus an das DataBaseQuery1. Dieser Wert wird im SQL-Ausdruck des DatabaseQuery-Steuerelements verwendet:

Abb. 259: Die SQL-Abfrage, die die ListBox mit Daten belegt



Der SQL-Befehl „select * from airport where id='%1'" enthält den Parameter %1, dem der RowTag-Wert des im Popup-Menu selektierten Eintrags zugewiesen wird, also die ID der vom Anwender ausgewählten Stadt.

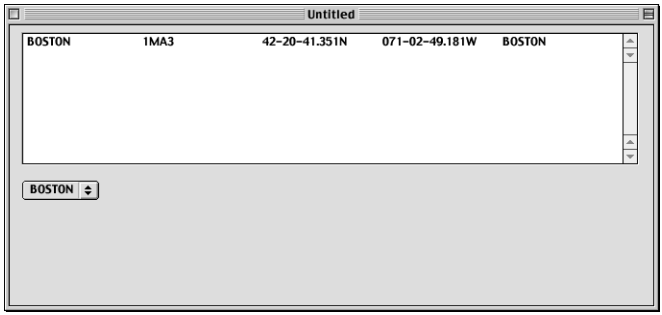
Mit anderen Worten: Diese Abfrage selektiert alle Spalten für die ausgewählte Stadt, verwendet dafür aber die unsichtbare Spalte an stelle des Namens der Stadt.

Die letzte Verknüpfung verknüpft die Ergebnisse des DatabaseQuery mit der ListBox:

```
Bind listBox1 with list data from DatabaseQuery1 results
```

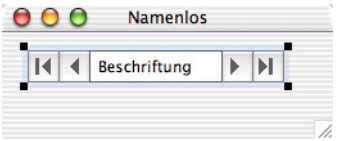
Ein Beispiel ist in Abbildung 260 zu sehen.

Abb. 260: Eine einfache Datenbank, die nur Objektverknüpfung verwendet

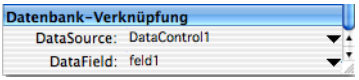


Das DataControl Steuerelement 

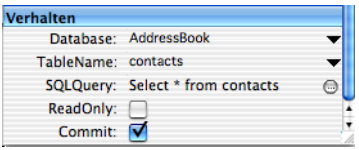
Mit dem DataControl-Steuerelement können Sie sehr einfach eine leistungsfähige Eingabemaske für eine Datenbanktabelle erzeugen. Das DataControl-Steuerelement ist ein einzelnes Objekt, das aus Navigationsknöpfen für die Datensätze (erster, letzter, nächster und voriger Datensatz) und einer Beschriftung besteht.



Ein DataControl-Steuerelement platzieren Sie auf einem Fenster, das EditFields, ListBoxes, Popup-Menüs oder StaticText-Steuerelemente verwendet, um Daten anzuzeigen und zu bearbeiten. Das DataControl-Steuerelement kann aber auch als Datenquelle für Klassen, die sich von diesen Steuerelementen ableiten, dienen. Diese Steuerelemente besitzen zwei Eigenschaften, DataSource und DataField, die nur in Verbindung mit einem DataControl verwendet werden. Diese werden im unteren Bereich des Eigenschaftenfensters aufgeführt:



Die *DataSource*-Eigenschaft enthält den Namen des DataControl-Objekts im Fenster. Das DataControl-Objekt wiederum ist über die Eigenschaften Database und TableName mit einer Datenbank-Tabelle verbunden:



In obiger Abbildung ist das DataControl mit der Tabelle „contacts“ der Datenbank „AddressBook“ verbunden. Sobald Sie die Datenbank „AddressBook“ ausgewählt haben, können Sie aus dem Popup unter TableName eine der in AddressBook vorhandenen Tabellen auswählen.

Die *DataField*-Eigenschaft enthält den Namen des Datenfeldes der contact-Tabelle, dessen Inhalt in diesem EditField anzeigt wird. Bei Verwendung der Datensatz-Navigationsknöpfe zeigt das DataControl automatisch den Wert dieses Feldes im EditField an.

Auf diese Weise verknüpfen Sie alle Steuerelemente der Eingabemaske mit dem DataControl und das DataControl mit einer Tabelle der Datenbank.

Folgende Aktionen können Steuerelementen mittels Objektverknüpfung zugewiesen werden:

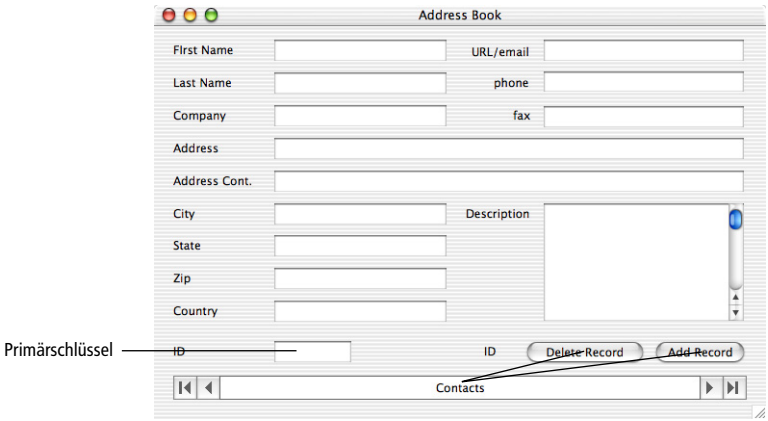
- Die Insert-Verknüpfung fügt den im Fenster gezeigten Datensatz in die Tabelle ein.
- Die Update-Verknüpfung speichert Änderungen an einem Datensatz in der Datenbank.
- Die New-Verknüpfung löscht die Inhalte aller Steuerelemente.
- Die Delete-Verknüpfung löscht den Datensatz aus der Tabelle.

Diese Operationen sind in folgender Tabelle zusammengefasst:

Tabelle 38. Objektverknüpfungen mit dem DataControl-Steuerelement

Quellobjekt	Verfügbare Objektverknüpfung
PushButton	Fügt Datensatz ein, wenn der PushButton gedrückt wird. Aktualisiert Datensatz, wenn der PushButton gedrückt wird. Löscht Datensatz, wenn der PushButton gedrückt wird. Neuer Datensatz, wenn der PushButton gedrückt wird.
BevelButton	Fügt Datensatz ein, wenn der BevelButton gedrückt wird. Aktualisiert Datensatz, wenn der BevelButton gedrückt wird. Löscht Datensatz, wenn der BevelButton gedrückt wird. Neuer Datensatz, wenn der BevelButton gedrückt wird.
RadioButton	Fügt Datensatz ein, wenn der RadioButton selektiert wird. Aktualisiert Datensatz, wenn der RadioButton selektiert wird. Löscht Datensatz, wenn der RadioButton selektiert wird. Neuer Datensatz, wenn der RadioButton selektiert wird. Fügt Datensatz ein, wenn der RadioButton deselektiert wird. Aktualisiert Datensatz, wenn der RadioButton deselektiert wird. Löscht Datensatz, wenn der RadioButton deselektiert wird. Neuer Datensatz, wenn der RadioButton deselektiert wird.
CheckBox	Fügt Datensatz ein, wenn die CheckBox angewählt wird. Aktualisiert Datensatz, wenn die CheckBox angewählt wird. Löscht Datensatz, wenn die CheckBox angewählt wird. Neuer Datensatz, wenn die CheckBox angewählt wird. Fügt Datensatz ein, wenn die CheckBox abgewählt wird. Aktualisiert Datensatz, wenn die CheckBox abgewählt wird. Löscht Datensatz, wenn die CheckBox abgewählt wird. Neuer Datensatz, wenn die CheckBox abgewählt wird. Aktiviert DataControl, wenn CheckBox angewählt ist.

Mittels Objektverknüpfung können Sie Knöpfe auf der Eingabemaske mit diesen Schlüsseloperationen belegen, ohne dafür Code schreiben zu müssen. Hier ein Beispielfenster, das zur Dateneingabe bereit ist:



Damit das DataControl korrekt mit der Datenquelle zusammen arbeiten kann, muss die Tabelle ein Primärschlüsselfeld besitzen. Ein Primärschlüssel ist eindeutig und darf nicht leer sein.

Wenn Sie als Datenquelle die REALdatabase verwenden, brauchen Sie keinen Primärschlüssel zu erzeugen, da jede Tabelle einer REALdatabase automatisch einen Primärschlüssel namens _rowID enthält. Das _rowID-Feld ist eine eindeutige, sequentiell aufsteigende Datensatznummer, die von der REALdatabase-Datenbank Engine automatisch vergeben wird. Wenn Sie eine andere Datenbank als Datenquelle verwenden, sollten Sie für den Primärschlüssel eine fortlaufende Nummer erzeugen. Legen Sie dazu ein EditField- oder StaticText-Steuerelement an, das mit dem Primärschlüsselfeld der Tabelle verknüpft ist und schreiben Sie folgenden Code in den Validate-Event des DataControls:

```
Function Validate(action as Integer) as Boolean
  Dim dbc as DatabaseCursor
  If action = me.kAddNew then
    dbc = Me.database.SQLSelect("Select Max(id) from " + me.tableName)
    idfield.text = Str(dbc.idxField(1).integerValue + 1)
  end if
End Function
```

Dieser Code geht davon aus, dass das Primärschlüsselfeld den Namen „id“ besitzt und mit dem Steuerelement „idfield“ verknüpft ist.

Programmieren eines Datenbank-Frontends

Um die Datenbankfunktionen von REALbasic auszureizen, kommen Sie nicht umhin, ein wenig Programmcode zu schreiben. Die Befehle, die im Datenbank-Teil der *Sprachreferenz* beschrieben werden, versorgen Sie mit allen notwendigen Werkzeugen, um anspruchsvolle Datenbank-Frontends zu entwickeln.

Auswahl einer Datenquelle

Die folgenden Klassen erlauben Ihnen die Auswahl der Datenbank-Backends, die in ihrer Applikation verwendet werden sollen. Mit Ausnahme der REAL-Datenbank setzt jede Methode voraus, dass das passende REALbasic-Plugin im Plugins-Ordner installiert ist. Alle Datenbank-Plugins werden zusammen mit REALbasic geliefert. Datenbank-Plugins werden häufiger aktualisiert als REALbasic selbst. Die neuesten Versionen finden Sie immer unter www.realsoftware.com.

Tabelle 39. Klassen zum Zugriff auf Datenquellen

Database-Unterklasse	Beschreibung
ADSP4DServer	Wird statt Open4DDatabasebyADSP verwendet.
MySQLDatabase	Unterstützung von MySQL-Datenbanken.
ODBCDatabase	Wird statt OpenODBCDatabase verwendet.
OpenBaseDatabase	Wird statt OpenOpenBaseDatabase verwendet.
OpenOracleDatabase	Sorgt für die Unterstützung von Oracle 8-9i unter Mac OS X.
PostgreSQLDatabase	Wird statt OpenPostgreSQLDatabase verwendet.
REALDatabase	Wird statt Open4DDatabasebyTCPIP verwendet.
TCPIP4DServer	Öffnet eine 4th Dimension Datenbank unter Verwendung von ADSP.

Frontbase kann ebenfalls verwendet werden. Unterstützung hierfür wird durch ein Plugin eines Drittanbieters bereitgestellt. Dieses Plugin ist frei verfügbar und befindet sich auf der REALbasic CD.

Da die oben genannten Klassen von der Database-Klasse abgeleitet wurden, stellen sie sowohl einen Standardbefehlssatz als auch von der konkreten Datenquelle abhängige Befehle zur Verfügung.

Weitere Informationen über die neuen Unterklassen der Database-Klasse finden Sie in der Sprachreferenz.

Wenn Sie eine Datenquelle öffnen, sollten Sie zunächst testen, ob die Verbindung erfolgreich aufgebaut werden konnte, bevor Sie Datenbankoperationen ausführen. Verwenden Sie dazu die Connect-Methode der Database-Klasse. Liefert

diese True zurück, konnte die Verbindung hergestellt werden. Wird False zurückgeliefert, können Sie anhand der ErrorMessage- und ErrorCode-Eigenschaften der Database-Klasse Rückschlüsse auf die Fehlerursache ziehen.

Der folgende Code öffnet eine 4D Server-Datenbank der Version 6.7 über TCP/IP:

```
Dim db as TCPIP4DServer67
db = New TCPIP4DServer67
db.host = "127.0.0.1"
db.username = "me"
db.password = "test"
If db.connect() then
//an dieser Stelle können Sie Änderungen an Ihrer Datenbank vornehmen
else
  MsgBox "Connection failure!"
end if
```

Mit Ausnahme von OpenCSVCursor liefern diese Methoden ein Objekt vom Typ Database zurück. Tabelle 40 zeigt die Methoden der Database-Klasse.

Tabelle 40. Methoden der Database-Klasse

Name	Parameter	Beschreibung
Close		Schließt die Datenbank.
Commit		Sichert Änderungen von Datensätzen in der Datenbank. Wenn Sie die Anwendung beenden, nachdem Sie Änderungen an einem RecordSet vorgenommen haben, führt REALbasic von sich aus den Commit-Befehl aus. Verwenden Sie Commit und Rollback, um Transaktionen zu verarbeiten.
Connect		Stellt die Verbindung zum Datenbank-Server her und öffnet die Datenbank für den Zugriff. Liefert ein Boolean. Bevor Sie mit Datenbankoperationen fortfahren, sollten Sie testen, ob der Connect-Befehl True zurückliefert.
FieldSchema	Tabellenname als String	Liefert ein RecordSet mit Informationen zu allen Feldern in der Tabelle.
InsertRecord	Tabellenname als String, Daten als DatabaseRecord	Fügt „Daten" in die letzte Zeile von „Tabellenname" ein.
Property	Name als String	Gibt die Datenbank-Eigenschaft „Name" der Datenquelle aus. Wird derzeit nur für 4D Server unterstützt.
Rollback		Bricht Transaktionen ab, ohne die Änderungen in der Datenbank abzuspeichern.
SQLSelect	SelectString als String	SQL Select-Befehl. Liefert ein RecordSet zurück.
SQLExecute	ExecuteString als String	Ein auszuführender SQL-Befehl. Wird verwendet, um SQL-Befehle auszuführen, die kein RecordSet zurückliefern.

Die Methode SQLSelect kann an Stelle eines DatabaseQuery-Steuerelements verwendet werden, um Abfragen an die Datenbank zu richten. Die SQLExecute-Methode kann statt des Datenbank-Schema-Dialogs verwendet werden, um eine Datenbankstruktur zu erzeugen (und auch viele andere Operationen ausführen). Beispielsweise kann folgender Befehl verwendet werden, um eine Tabelle zu erzeugen:

```
Dim db as Database
.
.
db.SQLExecute("create table authors(au_id integer not null, au_fname varchar, au_lname varchar,
address varchar, city varchar), state varchar, zip varchar, phone varchar, primary key (id)")
```

Editieren von Datensätzen

Die SQLSelect-Methode liefert ein Objekt des Typs RecordSet. In der Terminologie von SQL entspricht das einem Datenbank-Cursor. Frühere REALbasic-Versionen haben für diesen Zweck von der DatabaseCursor-Klasse Gebrauch gemacht. Funktionen, die vorher einen Datenbank-Cursor ausgegeben haben, liefern jetzt ein RecordSet.

RecordSets enthalten Datensätze, die das Selektionskriterium erfüllen, das durch die WHERE-Bedingung des SELECT-Befehls vorgegeben wurde. Außerdem werden die im RecordSet enthaltenen Datensätze in Multi-User-Umgebungen vor Änderungen durch andere Anwender geschützt. Die Datensätze eines RecordSets können angezeigt oder bearbeitet werden. Mehrere Datensätze können nur nacheinander, nicht gleichzeitig bearbeitet werden. Die Eigenschaften und Methoden des RecordSet-Objekts können Sie in der Sprachreferenz nachschlagen.

Wenn ein SQL-Befehl ein RecordSet zurückliefert, hat der Anwender das „Recht", die Datensätze des RecordSets exklusiv anzusprechen. Wenn das RecordSet mehrere Datensätze enthält, können die Eigenschaften und Methoden des RecordSets verwendet werden, um seine einzelnen Zeilen und Spalten anzusprechen.

Nachdem eine Abfrage mit der SQLSelect-Methode durchgeführt wurde, enthält das RecordSet einen Zeiger auf den aktuellen Datensatz (normalerweise auf den ersten Datensatz im Recordset). Sie können die Eigenschaften BOF (Beginning Of File) und EOF (End Of File) der RecordSet-Klasse verwenden, um zu ermitteln, ob sich der aktuelle Datensatzzeiger vor dem ersten oder hinter dem letzten Datensatz befindet. So können Sie feststellen, ob die SQLSelect-Methode überhaupt Datensätze gefunden hat. Wenn Ihre Abfrage keine Datensätze gefunden hat, sind sowohl BOF als auch EOF True, da der aktuelle Datensatzzeiger auf nichts zeigt.

Wenn Ihre Abfrage mehr als einen Datensatz findet, sind sowohl BOF als auch EOF False, da der aktuelle Datensatzzeiger auf den ersten Datensatz zeigt und nicht auf den Anfang oder das Ende der Datei. Wenn Ihre Abfrage zum Beispiel drei Datensätze gefunden hat, würde das RecordSet folgendermaßen aussehen:

```
BOF
Record1 (der aktuelle Datensatzzeiger zeigt auf diesen Datensatz)
Record2
Record3
EOF
```

Wenn der Datensatzzeiger auf einen Datensatz verweist, können Sie diesen mit den Edit- und Update-Methoden modifizieren bzw. mit der DeleteRecord-Methode löschen. Um den Datensatz zu modifizieren, rufen Sie zunächst die Edit-Methode auf und führen dann die Änderungen aus.

Um den Wert eines einzelnen Tabellenfeldes zu erfahren oder diesen zu setzen, verwenden Sie die Value-Eigenschaft der DatabaseField-Klasse. Da die Value-Eigenschaft ein Variant ist, können Sie sie mit jedem beliebigen Feld verwenden. Um den Wert eines Feldes auf Nil zu setzen, weisen Sie der Value-Eigenschaft dieses Feldes den Wert Null zu.

Sind die Änderungen an einem Datensatz abgeschlossen, rufen Sie die Update-Methode auf, um das RecordSet zu aktualisieren. Sind die Änderungen am RecordSet abgeschlossen, rufen Sie die Commit-Methode des Database-Objekts auf, um die Modifikationen in der Datenbank zu speichern. Sollen die Änderungen hingegen doch nicht gespeichert werden, müssen Sie Rollback aufrufen.

Die EOF-Eigenschaft nimmt den Wert True an, wenn Sie versuchen, den Datensatzzeiger über den letzten Datensatz hinaus zu bewegen

Wenn Sie nicht explizit Commit aufrufen, wird REALbasic dies implizit beim Schließen der Datenbank erledigen.

Datensätze einfügen

Ein neuer Datensatz wird unter Verwendung der InsertRecord-Methode des Database-Objekts eingefügt. Diese Methode hat zwei Parameter: Das Database-Objekt und ein Objekt vom Typ DatabaseRecord.

Tabelle 41. Eigenschaften von DatabaseRecord

Name	Typ	Beschreibung
Column	Name als String	Spalte der aktuellen Tabelle.

Der folgende Code fügt in der „Mitarbeiter"-Tabelle einen Datensatz ein:

```
dim db as Database
.
dim rec as DatabaseRecord
rec = New DatabaseRecord
.
rec.Column("au_id") = "09"
rec.Column("au_fname") = "Oscar"
rec.Column("au_lame")="Wilde"
db.InsertRecord("authors",rec)
```

Weitere Informationen und Beispiele finden Sie in der Sprachreferenz unter RecordSet, DatabaseField und bei den Datenbankklassen.

Kapitel 11 Debugging Ihres Codes

Wäre es nicht schön, wenn jede Programmzeile sich so verhalten würde, wie Sie es möchten? Und das auch noch ohne einen einzigen Fehler? Falls sich Ihre Programme einmal nicht so verhalten, bietet Ihnen REALbasic Werkzeuge, um den Fehlern auf die Schliche zu kommen und sie zu beheben.

Inhalt

- Was ist „Debugging“?
- Der Debugger und Breakpoints
- Die Ausführung von Methoden verfolgen
- Beobachten von Variablen und Eigenschaften
- Unterbrechen des Programms
- Abfangen von Runtime Exception Fehlern
- Remote Debugging

Was ist „Debugging“?

Debugging bedeutet, sowohl logische als auch Syntax-Fehler aus dem Programmcode zu entfernen. Fehler im Programmcode werden oft auch als „bugs“ (engl. für „Käfer“) bezeichnet. Möglicherweise fragen Sie sich, warum Fehler so genannt werden. In der Steinzeit der Computergeschichte, den 40er Jahren des 20. Jahrhunderts, baute man Computer, die eine ganze Lagerhalle ausfüllten. Wichtige Bauelemente der Rechner waren damals Vakuum-Röhren. Das Licht dieser Vakuum-Röhren zog auch Motten und Käfer an, die dann im Computer herumschlichen und dort Kurzschlüsse an den Röhren auslösten. Daraufhin mussten Techniker in den Computer gehen (!) und die Käfer entfernen, damit der Computer wieder funktionierte. Da es sich damals um Regierungsprojekte handelte, musste alles protokolliert werden, so dass die Techniker ins Protokoll „debugging computer“ („Käfer entfernen“) schrieben. Ende der Geschichtsstunde.

Debugging ist ein Teil des Programmierens. Es ist der Teil, den die meisten Programmierer am wenigsten mögen. REALbasic macht Ihnen die Fehlersuche glücklicherweise einfach. REALbasic enthält einen Debugger, der Ihnen in speziellen Fenstern zeigt, was gerade schiefgeht.

Logische Fehler

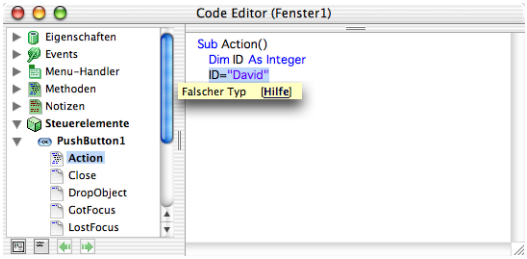
Dies sind Fehler im logischen Programmablauf. Sie wissen, dass Sie so einen Fehler gefunden haben, wenn Ihr Programmcode „etwas“ macht, aber „etwas anderes“ hätte machen sollen. Der eingebaute Debugger von REALbasic hilft Ihnen bei der Suche, indem Sie jede Zeile Ihres Programmcodes einzeln ausführen und die Reaktionen darauf beobachten können.

Syntax-Fehler

Dies sind Fehler, bei denen Sie sich bei der Eingabe des Namens einer Klasse, einer Eigenschaft, Variable oder Methode vertippt haben. Es kann auch sein, dass Sie probiert haben, zwei Variablen zusammen zu verwenden, die nicht zusammenarbeiten können. Wenn Sie beispielsweise einer Integer-Variable einen String-Wert zuweisen möchten, bekommen Sie einen „Type Mismatch error“, da es sich um zwei verschiedene Datentypen handelt. REALbasic macht es einfach, Syntax-Fehlern auf die Spur zu kommen. Sobald Sie im Menü **Debug/Programm starten** (⌘-R oder Strg-R) ausführen,

prüft REALbasic Ihren Programmcode auf Syntaxfehler und bemängelt diese, statt das Projekt zu compilieren. Sie müssen diese Fehler beheben, bevor Ihr Projekt ablaufen kann. Abbildung 261 zeigt ein Beispiel eines solchen Fehlers.

Abb. 261: Eine Syntax-Fehlermeldung

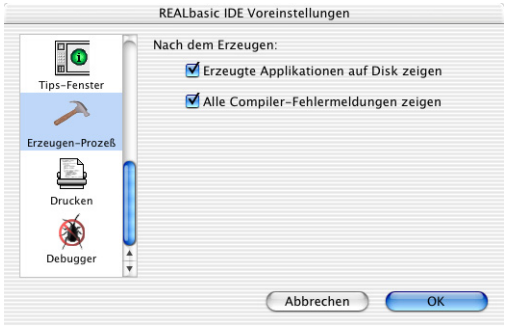


REAlbasic hebt den ersten Fehler hervor, den der Compiler findet. REALbasic markiert die fehlerhafte Zeile und zeigt ein Tippsfenster mit der Fehlermeldung an. Der oben gezeigte Fehler trat auf, weil die Variable ID als Integer definiert ist, der Programmierer aber versucht hat, ihr einen String zuzuweisen.

Rechts neben der Fehlermeldung befindet sich ein „Hilfe“-Knopf. Wenn Sie den „Hilfe“-Knopf anklicken, wird der entsprechende Eintrag der Online-Hilfe mit einer Beschreibung der Fehlermeldung aufgerufen.

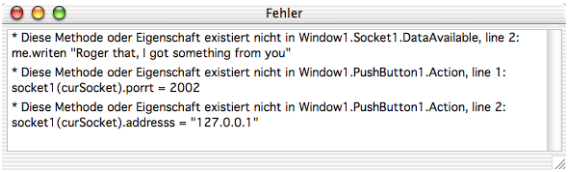
Bei größeren Projekten ist es meist sinnvoll, eine Liste mit allen Syntax-Fehlern auf einmal zu erhalten. Klicken Sie dazu auf **REALbasic/Einstellungen** bzw. **Bearbeiten/Einstellungen** und aktivieren auf der Dialogseite „Erzeugen-Prozess“ die Option „Alle Compiler-Fehlermeldungen zeigen“.

Abb. 262: „Alle Compiler-Fehlermeldungen zeigen“ im Einstellungen-Dialog



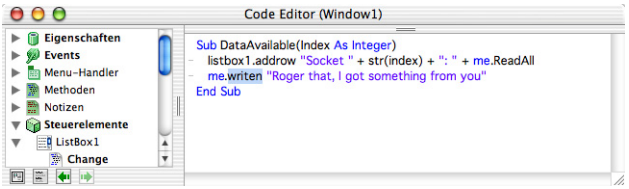
Dann werden alle Fehler in einem Fehlerfenster angezeigt und nicht mehr als Tipps im Code-Editor.

Abb. 263: Das Fehlerfenster

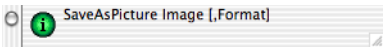


Jeder Eintrag enthält eine Fehlermeldung und die Fehlerposition. Um die fehlerhafte Zeile im Code anzuspringen, doppelklicken Sie auf die Fehlermeldung oder selektieren diese und drücken Return. REALbasic öffnet daraufhin den Code-Editor und zeigt den Fehler an.

Abb. 264: Der erste Fehler aus dem Fehlerfenster im Code-Editor



Um Syntaxfehler zu vermeiden, können Sie während des Codierens das Tippsfenster zu Rate ziehen und die Syntax für jedes Element überprüfen. Folgender Eintrag erscheint z.B. im Tippsfenster, wenn Sie der FolderItem-Klasse die SaveAs-Picture-Methode übergeben:

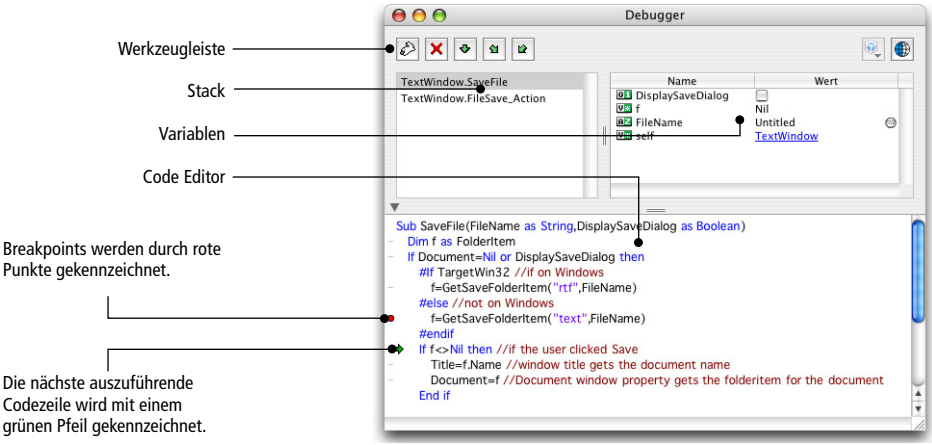


Sie werden also darauf hingewiesen, dass die Methode das zu speichernde Bild und gegebenenfalls ein Format erwartet. Die möglichen Format-Codes finden Sie in der Sprachreferenz.

Der Debugger und Breakpoints

Im Debugger-Fenster können Sie Ihren Code Zeile für Zeile ausführen und dabei die Werte der Variablen beobachten. Der Debugger markiert die Programmzeile, die als nächstes ausgeführt wird, mit einem grünen Pfeil am Zeilenanfang. Wenn der Debugger aktiv ist, werden alle Einträge des Debug-Menüs aktiviert.

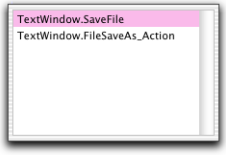
Abb. 265: Das Debugger-Fenster



Das Debugger-Fenster ist in drei Bereiche unterteilt. Der Bereich des Code-Editors zeigt die Methode, die momentan ausgeführt wird. Die Ausführung wurde in unserem Beispiel in der Zeile direkt unterhalb des Breakpoints ausgesetzt. Der rote Punkt markiert einen Breakpoint. Wenn der Debugger nach dem Starten des Programms zum ersten Mal erscheint, weil Sie einen Breakpoint gesetzt haben, befindet sich der grüne Pfeil direkt am Breakpoint.

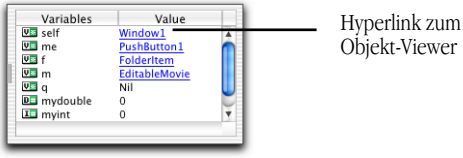
Das Stack-Feld (aus dem engl.: „to stack“, dt. stapeln) enthält die Namen der aktuellen Methode und aller Methoden, die zum Aufruf der aktuellen Methode geführt haben. Sie werden in der Reihenfolge angezeigt, in der sie aufgerufen wurden. Im Stack-Feld können Sie also überprüfen, ob eine Methode auch tatsächlich dann aufgerufen wird, wenn Sie es wünschen. Mit einem Klick auf die jeweilige Methode können Sie deren Code einsehen.

Abb. 266: Das Stack-Feld zeigt die Reihenfolge an, in der die Methoden aufgerufen werden



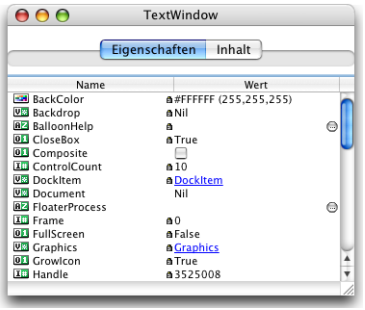
Das Variablen-Feld zeigt für die Methode, die gerade im Code-Editor dargestellt wird, eine Liste der lokalen Variablen samt deren Werte (falls vorhanden). Der Datentyp jeder Variablen wird mit einem kleinen Symbol angezeigt.

Abb. 267: Das Variablenfenster



Alle Objekte der Methode werden als Hyperlinks dargestellt. Beim Klick auf einen Hyperlink öffnet sich der „Object-Viewer“. Dieser enthält eine Liste mit den aktuellen Werten der Eigenschaften des angeklickten Objekts.

Abb. 268: Der Object-Viewer



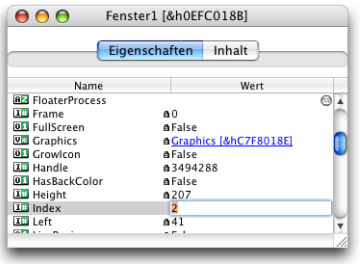
Der Vorteil von Object-Viewer und Variablen-Feld besteht in deren Interaktivität. Während Sie im Debugger den Code Zeile für Zeile ausführen, werden in beiden Feldern die Werte in Echtzeit aktualisiert. Auf diese Weise können Sie herausfinden, ob ein bestimmter Wert (oder vielleicht auch ein fehlender Wert) das Problem verursacht.

Ein Object-Viewer für ein Fenster zeigt zwei Karteikarten an: Eigenschaften und Inhalt. Die Karteikarte Eigenschaften zeigt die aktuellen Werte der Eigenschaften des Fensters. Die Karteikarte Inhalt enthält Hyperlinks auf die Steuerelemente des Fensters.

Anders als bei Fenstern haben die Object-Viewer der meisten anderen Steuerelemente nur die Karteikarte mit den Eigenschaften.

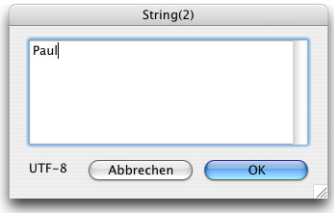
Die Karteikarte „Eigenschaften" des Object-Viewers

Die Eigenschaften-Karteikarte zeigt alle Eigenschaften eines Objekts und deren aktuelle Werte. Mit einem Schloss markierte Eigenschaften können nur ausgelesen werden. Die anderen Eigenschaften können Sie im Object-Viewer bearbeiten. Selektieren Sie dazu die entsprechende Zelle der Wertespalte. Ändern Sie den Inhalt der Zelle und drücken Sie die Tab-Taste oder selektieren Sie eine andere Zelle, um die Änderung zu übernehmen.

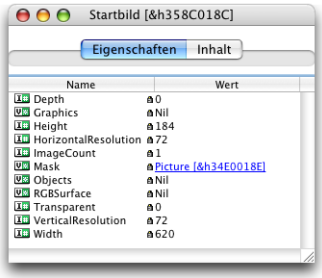


Auch Arrays können im Debugger bearbeitet werden. In der Wertespalte wird dann ein Hyperlink angezeigt, der einen eigenen Dialog öffnet. In diesem Dialog können Sie die Elemente des Arrays bearbeiten. In der folgenden Abbildung handelt es sich bei der Eigenschaft „Namen" um ein Array mit drei Elementen.

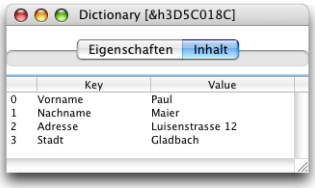
Enthält eine Eigenschaft Text, wird dieser in der Wertespalte angezeigt. Falls dort nicht genügend Platz zur Verfügung steht, können Sie den kleinen Knopf am Zeilenende anklicken und damit den String-Viewer öffnen. Dieser zeigt nicht nur den vollständigen Text, sondern auch dessen Codierung an.



Bei Eigenschaften vom Typ „Picture“ zeigt der Object-Viewer die zwei Karteikarten „Eigenschaften“ und „Inhalt“ an. Unter „Inhalt“ kann man sich das Bild ansehen.



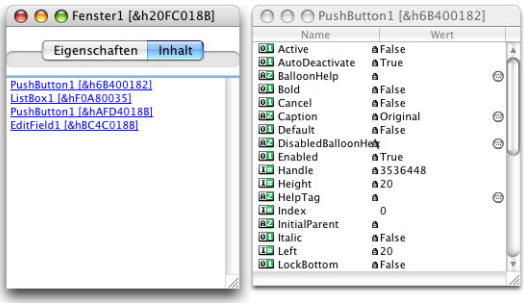
Handelt es sich bei der Eigenschaft um ein Dictionary, besitzt der Object-Viewer ebenfalls zwei Karteikarten. Unter „Inhalt“ werden die Schlüssel-Wert-Paare angezeigt, die sich im Dictionary befinden.



Eigenschaften, die Objekte sind, werden als Hyperlinks angezeigt und öffnen jeweils einen Object-Viewer. Es können mehrere Object-Viewer gleichzeitig geöffnet sein.

Die Karteikarte „Inhalt“ des Object Viewers

Die Karteikarte „Inhalt“ des Object-Viewers für ein Fenster zeigt die Steuerelemente an, die sich in dem Fenster befinden. Die folgende Abbildung zeigt den Object-Viewer für PushButton1.



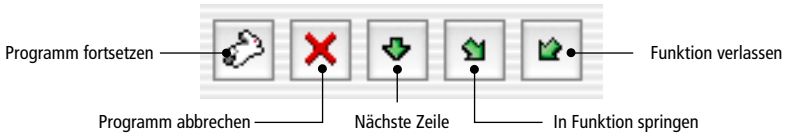
Jede Verknüpfung öffnet einen Object-Viewer für das Objekt. Dies funktioniert bis zur letzten Ebene von Objekten.

Wenn ein Fenster beispielsweise ein ImageWell-Objekt enthält, wird dieses im Object-Viewer des Fensters als Hyperlink angezeigt. Enthält das ImageWell-Objekt ein Bild, erscheint dieses im Object-Viewer des ImageWell-Objekts ebenfalls als Hyperlink.

Die Werkzeugleiste des Debuggers

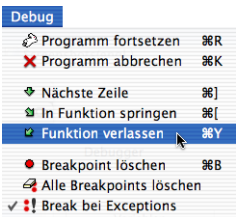
Die Werkzeugleiste des Debuggers enthält fünf Knöpfe, die einen direkten Zugriff auf Funktionen des **Debug**-Menüs gestatten. Mit den Knöpfen können Sie die Ausführung Ihres Programms steuern. Es ist möglich, den Programmcode Zeile für Zeile auszuführen.

Abb. 269: Werkzeugleiste des Debuggers



Das **Debug**-Menü enthält die gleichen Funktionen wie die Werkzeugleiste des Debuggers.

Abb. 270: Das Debug-Menü bei aktivem Debugger



- **Programm fortsetzen** (in der IDE: **Programm starten**): Setzt die Ausführung des Codes bis zum nächsten Breakpoint fort. Mit diesem Befehl verlassen Sie das Debugger-Fenster.
- **Programm abbrechen**: Bricht die Ausführung des Programmcodes ab und bringt Sie zurück zur REALbasic Entwicklungsumgebung.
- **Nächste Zeile**: Führt die aktuelle – mit einem grünen Pfeil markierte – Programmzeile aus. Springt zur nächsten Zeile. Wenn die aktuelle Zeile eine Methode beinhaltet, führt der Debugger diese Methode auf einmal aus, arbeitet sie also nicht zeilenweise ab.
- **In Funktion springen**: Führt die aktuelle Zeile aus und springt auf die nächste. Falls die aktuelle Zeile einen Methodenaufruf enthält, zeigt der Debugger deren Code an und erlaubt es, diesen ebenfalls zeilenweise abzuarbeiten.
- **Funktion verlassen**: Die aktuelle Methode wird bis zum Ende ausgeführt, ohne dass jede Zeile einzeln bestätigt werden muss. Das ist besonders dann sehr praktisch, wenn Sie vorher mit dem Befehl **In Funktion springen** in die Methode gesprungen sind und diese nun wieder verlassen wollen.

Wann wird der Compiler-Lauf und die Programmausführung abgebrochen?

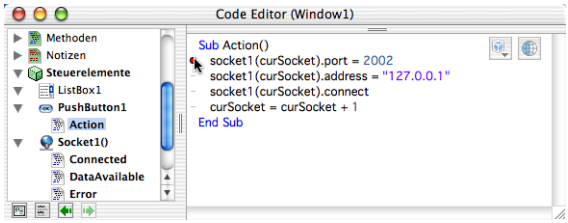
Sie haben einen Syntaxfehler in Ihrem Code

Wenn Sie ihr Projekt über **Debug/Programm starten** (⌘-R oder Strg-R) oder **Datei/Applikation erzeugen** ausführen oder übersetzen möchten, prüft REALbasic den Programmcode auf Syntaxfehler. Wird ein Fehler gefunden, so wird der Übersetzungsvorgang abgebrochen und die Fehlerposition im Code-Editor gezeigt.

Sie haben einen Breakpoint gesetzt

Sie können einen *Breakpoint* in eine Programmzeile setzen, bei deren Erreichen REALbasic das Programm anhalten und den Debugger öffnen soll. Die Zeilen, in die Sie einen Breakpoint setzen können, werden im Code-Editor mit einem Bindestrich am Zeilenanfang markiert. Zum Setzen eines Breakpoints klicken Sie einfach auf den Bindestrich.

Abb. 271: Setzen eines Breakpoints



Breakpoints werden als rote Punkte angezeigt und bleiben so lange erhalten, bis sie entfernt werden. Einen Breakpoint können Sie entfernen, indem Sie auf den roten Punkt klicken. Über den Menübefehl **Debug/Alle Breakpoints löschen** können Sie alle Breakpoints auf einmal entfernen.

In einem Projekt gesetzte Breakpoints haben keine Auswirkungen auf mittels **Ablage/Applikation erzeugen** generierte Stand-Alone-Anwendungen.

Sie haben ctrl-C oder eine äquivalente Tastenkombination gedrückt

Sie können die Ausführung eines mit **Debug/Programm starten** gestarteten Programms unterbrechen und den Debugger aufrufen, indem Sie die Tastenkombination ctrl-C (Windows ctrl-R oder F5) drücken. Dies ist besonders dann wichtig, wenn Sie entdecken, dass sich Ihr Programm in einer Endlosschleife befindet.

Die Tastenkombination ⌘-Shift-Punkt kann dazu nicht mehr verwendet werden.

Alternativ gelangen Sie zur Entwicklungsumgebung zurück, indem Sie ein Fenster der Entwicklungsumgebung anklicken.

Die Ausführung von Methoden verfolgen

Wenn sich Ihr Programm nicht wie erwartet verhält oder Sie nicht sicher sind, welche Methoden wann ausgeführt werden, kann es nützlich sein, die Ausführung jeder einzelnen Programmzeile zu beobachten. Der Debugger bietet Ihnen diese Möglichkeit. Wenn der Debugger aufgerufen wird, zeigt der grüne Pfeil an, welche Zeile als nächstes ausgeführt wird. Jetzt haben Sie drei Möglichkeiten, das Programm schrittweise abzuarbeiten.

Nächste Zeile

Der Befehl **Nächste Zeile** führt die aktuelle Zeile aus stoppt den Programmablauf in der nächsten Zeile. Wenn die aktuelle Zeile den Aufruf einer Methode enthält, führt der Debugger diese Methode auf einmal aus, also *nicht* zeilenweise, und springt zur nächsten Zeile der aktuellen Methode. Nehmen Sie folgenden Code als Beispiel:

```
EditField1.SelBold=True
EditField1.Text=ToFrench(EditField1.Text)
EditField.SelBold=False
```

Nehmen wir an, dass die Methode „ToFrench“ von Englisch nach Französisch übersetzt. Wenn Sie den Code mit dem Befehl **Nächste Zeile** schrittweise abarbeiten, wird die zweite Zeile ausgeführt, ohne dass der Debugger den Code der Methode „ToFrench“ anzeigt. Er führt „ToFrench“ einfach aus und fährt mit der nächsten Zeile fort.

In Funktion springen

Der Befehl **In Funktion springen** führt die aktuelle Zeile aus und springt in die nächste Zeile. Enthält die aktuelle Zeile allerdings den Aufruf einer Methode, verzweigt der Debugger in diese Methode und führt deren Code zeilenweise aus. Wenn die Methode ausgeführt wurde, arbeitet der Debugger die aufrufende Methode weiter ab.

Funktion verlassen

Die aktuelle Methode wird bis zum Ende ausgeführt, ohne dass jede Zeile einzeln bestätigt werden muss. Das ist besonders dann sehr praktisch, wenn Sie vorher mit dem Befehl **In Funktion springen** in die Methode gesprungen sind und diese nun wieder verlassen wollen.

Die Ausführung von Methoden mit dem Stack verfolgen

Eine Methode oder ein Event-Handler kann eine andere Methode oder einen anderen Event-Handler aufrufen, der wiederum weitere aufrufen kann. Da man dies beliebig fortführen kann, braucht man eine Möglichkeit, die Reihenfolge zu protokollieren, in der die Methoden aufgerufen wurden. Das Stack-Feld leistet genau dies. Wenn Code ausgeführt wird (z.B. wenn ein PushButton angeklickt wird), zeigt das Stack-Feld den Event-Handler „Action" des Buttons. Wenn der Event-Handler „Action" eine Methode aufruft, wird diese Methode am Anfang der Stack-Liste eingefügt. Wenn diese Methode eine weitere Methode aufruft, wird wiederum diese an den Anfang der Liste gesetzt. Wenn die aktuelle Methode beendet wurde, wird sie aus der Liste entfernt. Abbildung 266 zeigt das Stack-Feld mit einigen Methoden.

Indem Sie eine auf dem Stack liegende Methode anklicken, können Sie sich deren Quelltext im Code-Bereich des Debugger-Fensters anschauen.

Beachten Sie: Je größer die Liste im Stack wird, desto mehr Speicher benötigt Ihr Programm. Falls Sie deshalb Probleme mit dem Speicherplatz bekommen sollten, müssen Sie versuchen, weniger Methoden-Aufrufe ineinander zu verschachteln und weniger lokale Variablen zu verwenden.

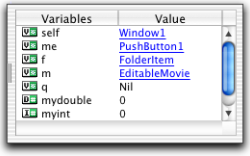
Beobachten von Variablen und Eigenschaften

Im Debugger können Sie die Werte von Variablen, Objekten und Eigenschaften verfolgen, während der Code ausgeführt wird. Dafür gibt es das Variablen-Feld. Dieses Fenster zeigt lokale Variablen, Parameter, das aktuelle Objekt und dessen übergeordnete Klasse an. Es zeigt auch globale Eigenschaften von Modulen und die Unterklasse, falls es eine gibt.

Lokale Werte

In Abbildung 272 wird gerade der „Action"-Event-Handler eines PushButtons ausgeführt. Die lokale Variable Self verweist auf das übergeordnete Fenster. Me verweist auf das Objekt, dessen Code ausgeführt wird (in diesem Fall den PushButton). Die Variablen f und m sind vom Typ FolderItem bzw. EditableMovie.

Abb. 272: Das Variablen-Feld



Für Objekte zeigt die „Wert"-Spalte ein Hyperlink. Mit einem Klick auf den Hyperlink öffnen Sie den Object-Viewer. Dieser zeigt die aktuellen Werte für alle Eigenschaften des Objekts an. Sie können so viele Object-Viewer öffnen, wie Sie benötigen. Object-Viewer können geöffnet bleiben, während Sie den Code abarbeiten, und werden automatisch aktuali-

siert, wenn sich eine Eigenschaft ändert. Object-Viewer sind separate Fenster, deren Größe Sie anpassen können, um auch lange Textzeilen, wie z.B. die AbsolutPath-Eigenschaft eines FolderItems, problemlos lesen zu können.

Negative Zahlen werden im Object-Viewer in roter Schrift dargestellt. Hyperlinks sind blau und unterstrichen.

Beachten Sie: Derzeit unterstützt der Object-Viewer nur eindimensionale Arrays.

Parameter

Im Variablen-Feld werden auch die Parameter angezeigt, die an eine Methode übergeben wurden.

Globale Werte

Am rechten Rand der Werkzeugleiste des Debugger-Fensters befinden sich zwei weitere Icons.

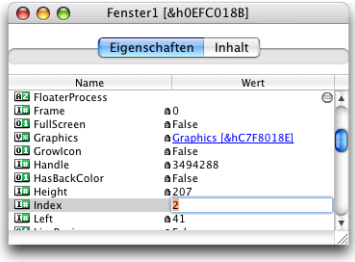
Das erste gestattet den Zugriff auf Module. Besteht Ihr Projekt aus zwei oder mehr Modulen, werden diese in einem Popup-Menü aufgelistet, sobald Sie auf das Modulsymbol klicken. Damit haben Sie Zugriff auf geschützte Eigenschaften des Moduls.

Das zweite Symbol ermöglicht den Zugriff auf globale Variablen (ungeschützte Eigenschaften von Modulen).

Objekt-IDs

Der Debugger kann die vom Compiler verwendeten Objekt-IDs anzeigen. Diese sind allerdings zum Debuggen eines REALbasic-Projekts nicht nötig. Deshalb werden sie per Voreinstellung nicht angezeigt. In den REALbasic-**Einstellungen** können Sie unter „Debugger“ die Option „Zeige Objekt-IDs in der Variablen-Liste“ aktivieren.

Im Variablen-Feld des Debuggers wird dann die ID eines Objekts in eckigen Klammern hinter dessen Typ oder Klasse angezeigt. In der Abbildung sehen Sie eine solche ID für die Eigenschaft „Graphics“.



Starten und Stoppen des Projekts

Wenn Sie Ihr Programm aus der REALbasic-Entwicklungsumgebung mittels **Debug/Programm starten** gestartet haben, können Sie die Programmabarbeitung jederzeit anhalten und in die Entwicklungsumgebung wechseln, indem Sie auf ein Fenster der Entwicklungsumgebung klicken.

Soll die Programmabarbeitung fortgesetzt werden, wählen Sie erneut den Menüpunkt **Debug/Programm starten** (⌘-R bzw. Strg-R). Soll das Programm nicht fortgesetzt werden, rufen Sie den Menüpunkt **Debug/Kill** (⌘-K bzw. Strg-K) auf.

Runtime Exception Fehler

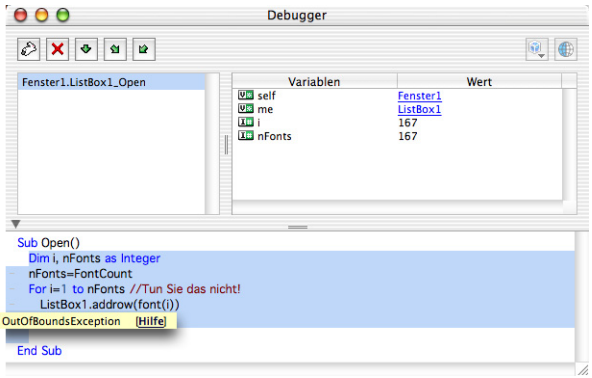
Laufzeitfehler in der Runtime-Umgebung der Entwicklungsumgebung

Im Gegensatz zu Syntaxfehlern, die REALbasic bereits während des Compilierens entdecken kann, treten Laufzeitfehler erst während der Ausführung eines Programms auf. Es ist also möglich, syntaktisch korrekten Programmcode zu schreiben, der beim Ausführen des Programms zu einem Fehler führt.

Wenn Sie Ihr Projekt mit **Debug/Programm starten** aus der Entwicklungsumgebung starten und während der Programmabarbeitung ein Laufzeitfehler auftritt, wird das Programm abgebrochen und die Fehlerposition im Code-Editor markiert.

Um Laufzeitfehler mit dem Debugger aufzuspüren, sollten Sie **Debug/Break bei Exceptions** aktivieren. Dann meldet sich der Debugger, wenn ein Laufzeitfehler auftritt, und gibt Ihnen die Möglichkeit, nach der Fehlerursache zu suchen.

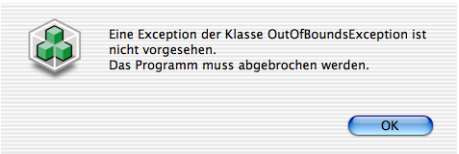
Abb. 273: Ein Runtime Exception Fehler im Debugger



Laufzeitfehler in der fertigen Anwendung

Wenn der Fehler in einer Stand-Alone-Applikation auftritt, erscheint eine Fehlermeldung, die den Typ des aufgetretenen Problems beschreibt. Hier ein Beispiel:

Abb. 274: Ein nicht abgefangener Runtime Exception Fehler in einer Stand-Alone-Applikation



Die Applikation wird beendet, sobald der Anwender die Fehlermeldung bestätigt.

Laufzeitfehler abfangen

REALbasic kann Laufzeitfehler erkennen und abfangen. Es gibt mehrere sogenannte Runtime Exceptions, die Sie in Ihrem Code erkennen und abfangen können, um zu verhindern, dass das Programm abstürzt. Diese Runtime Exceptions sind in folgender Tabelle aufgeführt:

Tabelle 42. Runtime Exception Fehler in REALbasic

Name	Beschreibung	Beispiel

IllegalCastException	Sie haben ein Objekt auf eine andere Klasse „gecastet" und ihm eine Nachricht geschickt, die seine tatsächliche Klasse nicht verarbeiten kann.	Einen BevelButton als PushButton verwenden:
		`Dim c as Control`
		`c=New BevelButton1`
		`PushButton(c).Push`

InvalidParentException	Sie haben versucht, über die Parent-Eigenschaft der Control-Klasse auf den Elternteil (Parent) eines Steuerelements zuzugreifen. Das Steuerelement befindet sich jedoch in einem anderen Fenster.	Dem Code selbst sieht man nichts an, nur die Benutzeroberfläche wäre fehlerhaft.

KeyNotFoundException	Sie haben versucht, auf einen Schlüssel zuzugreifen, der nicht Bestandteil eines Dictionary-Objekts ist.	Der Zähler nimmt einen Wert an, den es im Dictionary nicht gibt.
		`For i=1 to d.Count`
		`  ListBox1.Addrow d.value(i)`
		`Next`

KeyChainException	Eine Methode der KeyChain- oder KeyChainItem-Klasse konnte nicht ausgeführt werden.	Sie haben versucht, mehr als ein KeyChain-Element für dieselbe Applikation anzulegen.

NilObjectException	Es wurde versucht, auf ein nicht existierendes Objekt zuzugreifen.	`Zugriff auf ein Nil-FolderItem:`
		`Dim f as FolderItem`
		`f.Delete`

OLEException	Ein in Zusammenhang mit OLE stehender Laufzeitfehler ist aufgetreten.	

OutOfBoundsException	Es wurde versucht, einen Wert auszulesen oder zu schreiben, der außerhalb der Grenzen des Objekts oder des Datentyps liegt.	Zugriff auf ein Array-Element, das nicht existiert:
		`Dim a(3) as String`
		`a(4)="Hugo"`

OutOfMemoryException	Eine Aufgabe konnte nicht ausgeführt werden, da nicht ausreichend Speicher zur Verfügung stand.	Große Bitmaps belegen sehr viel Speicher.
RbScriptAlreadyRunning Exception	Der Benutzer hat versucht, ein laufendes Script oder dessen Kontext-Objekt zu ändern.	Vermeiden Sie dies!
RegExException	Sie haben ein ungültiges Suchmuster in einem regulären Ausdruck verwendet.	Fehlende „]" am Ende eines regulären Ausdrucks: `[aeiou`

Name	Beschreibung	Beispiel
RegistryAccessError Exception	Sie haben auf die Registrier–datenbank zugegriffen, ohne eine entsprechende Erlaubnis des Betriebssytems zu haben.	
ShellNotRunning Exception	Sie haben versucht, auf eine interaktive Shell zuzugreifen, nicht gestartet war.	Überprüfen Sie die IsRunning-Eigenschaft des Shell-Klasse-Objekts, bevor Sie darauf zugreifen.

StackOverflowException	Wenn Ihrer Applikation der Stack-Speicher ausgeht, tritt eine StackOverflowException auf.	Aufruf einer rekursiven Methode, bis der Stack überläuft:
		`Sub Square (i as Integer) as Integer`
		`  Return Square(i)`
		`End Sub`
		`Die Aufruf-Routine:`
		`Dim i as Integer`
		`i=Square(2)`

TypeMismatchException	Sie haben versucht, einem Objekt den falschen Datentyp zuzuweisen.	Sie versuchen einer Integer-Variablen einen Variant zuzuweisen, dem ursprünglich ein Bild zugewiesen wurde (das ins Projektfenster gezogen wurde).
		`Dim v as Variant`
		`Dim i as Integer`
		`v=REALlogo`
		`i=v`

UnsupportedFormat Exception	Sie verwenden einen String-Audruck, der keine Zahl ergibt, in einem Kontext, in dem das Ergebnis eine Zahl sein muss.	Sie übergeben einen String an die ColumnWidth-Eigenschaft einer ListBox, der keine Pixel- oder Prozentzahl ergibt. `ListBox1.ColumnWidths="50,50a"`

Eine Beschreibung jedes Fehlertyps samt Beispielen finden Sie in der Sprachreferenz.

Sie können mit einem Exception Block Laufzeitfehler abfangen und auf diese reagieren (z.B. informativere Warnboxen anzeigen) und verhindern, dass die Applikation abstürzt. Exception Blocks stehen im Programmcode immer am Ende einer Methode, da jede dem Exception-Schlüsselwort folgende Zeile als Teil des Exception Blocks angesehen wird.

Wenn ein Exception Block einen Laufzeitfehler abfängt, wird der Code im Exception Block ausgeführt und erlaubt Ihnen, Informationen über die Problemstelle und die Werte, die den Fehler ausgelöst haben, anzuzeigen.

Ein Exception Block ermöglicht es Ihnen, individuell auf Fehler zu reagieren. Damit haben Sie mehr Kontrolle darüber, wie mit einem Fehler umgegangen wird, als bei Aktivieren des **Break bei Exceptions**-Menüpunkts.

Die Exception-Anweisung erscheint im Code-Editor mit der gleichen Einrückung wie die Sub- oder Function-Zeile.

Sie können „Exception“ ohne Parameter verwenden, wenn Sie auf alle Laufzeitfehler identisch reagieren wollen:

```
Sub…
.
.
Exception
  MsgBox "Irgendwas Schlimmes ist passiert, ich weiß aber nicht, was."
End Sub
```

Die Syntax eines Exception Blocks lautet:

Exception *errorParameter* **As** *errorType*

errorParameter und *errorType* sind optional, errorType kann nicht ohne errorParameter verwendet werden.

Das oben gezeigte Beispiel bewahrt das Programm zwar vor einem Absturz, die Fehlermeldung sagt aber nichts über die aufgetretene Exception aus.

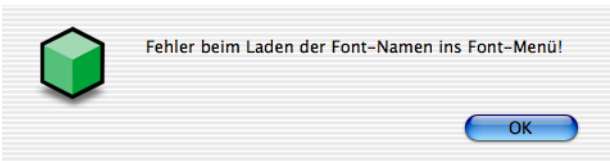
Mit einem If-Befehl im Exception Block können Sie *errorParameter* testen:

```
Sub…
.
.
Exception err
  If err IsA TypeMismatchException then
    MsgBox "Sie haben versucht, den Typ eines Objekts zu ändern!"
  elseif err IsA NilObjectException then
    MsgBox "Sie haben versucht, auf ein Nil-Objekt zuzugreifen!"
  End if
End Sub
```

Der OutOfBoundsException-Laufzeitfehler kann folgendermaßen abgefangen werden:

```
Sub Action()
  Dim i, nFonts as Integer
  nFonts=FontCount
    for i=1 to nFonts
      listBox1.addrow(Font(i))
    next
Exception err
  if err IsA OutOfBoundsException then
    msgBox "Fehler beim Laden der Font-Namen ins Font-Menü!"
  end if
End Sub
```

Wird dieser Code ausgeführt, erscheint folgende Dialogbox.:



Wenn der Anwender diese Warnbox bestätigt, wird das Programm nicht beendet.

Statt mehrfacher If-Anweisungen können Sie auch mehrere Exception Blocks verwenden, wobei jeder auf einen anderen Exception-Typ reagieren kann.

```
Sub…
.
.
Exception err as TypeMismatchException
  MsgBox "Sie haben versucht, den Type eines Objekts zu ändern!"
Exception err as NilObjectException
  MsgBox "Sie haben versucht, auf ein Nil-Objekt zuzugreifen!"
End Sub
```

Falls die Methode, in der ein Laufzeitfehler auftritt, bzw. deren aufrufende Methode keinen Exception Block besitzt, haben Sie eine letzte Chance, Laufzeitfehler abzufangen, bevor diese die Anwendung zum Absturz bringen. Dazu gibt es den UnhandledException-Event der Application-Klasse. Dieser Event wird ausgelöst, wenn irgendwo in der Anwendung ein Laufzeitfehler auftritt, der nicht von einem Exception Block abgefangen wurde. Der UnhandledException-Event erhält einen Parameter des Typs RuntimeException. So können Sie eine Funktion schreiben, die sich um die Exception kümmert. Aus Sicherheitsgründen könnten Sie eine allgemeine Routine schreiben, die alle in Tabelle 42 aufgeführten RuntimeExceptions abfängt.

Folgender Programmcode im UnhandledException-Handler des App-Objekts fängt alle Laufzeitfehler des Typs OutOfBoundsException ab:

```
Function UnhandledException(error as RuntimeException) as Boolean
  If error IsA OutOfBoundsException then
    MsgBox "Ein OutOfBounds Exception Fehler ist aufgetreten!"
  End if
  Return True
End Function
```

Diese Art, Laufzeitfehler abzufangen, liefert keinen eindeutigen Hinweis auf die genaue Fehlerursache, da alle OutOfBoundsExceptions der Anwendung abgefangen werden.

Remote Debugging

Mit Remote Debugging („ferngesteuertem Debuggen“) können Sie Ihr Projekt unter einem anderen Betriebssystem auf einem anderen Computer testen. Wenn Sie beispielsweise Ihre Applikation unter Mac OS X entwickeln und sie unter Windows testen wollen, wird eine Debug-Version der Applikation über das Netzwerk auf den Zielrechner geschickt und dort automatisch gestartet.

Remote Debugging ist nur in der Professional-Version von REALbasic verfügbar.

In den folgenden Ausführungen heißt der Computer, auf dem REALbasic läuft, „Entwicklermaschine“ und der Computer, auf dem die Applikation getestet wird, „Testmaschine“.

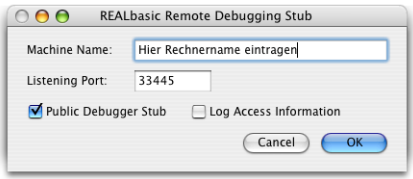
Remote Debugging benötigt ein Hilfsprogramm, das auf der Testmaschine gestartet werden muss. Dieses Programm heißt „Remote Debugger Stub“ und muss zunächst auf der Testmaschine installiert und konfiguriert werden. Danach können Sie auf der Entwicklermaschine das Remote Debugging über **Debug/Starte Remote-Debugging** starten.

Um Remote Debugger Stub zu konfigurieren, führen Sie folgende Schritte aus:

1. Kopieren Sie Remote Debugger Stub auf die Testmaschine und starten Sie es mit einem Doppelklick.

   Das Hauptfenster erscheint, in dem der Name und die IP-Adresse leer sind.

2. Wählen Sie **Edit/Options** (Windows und Linux) oder **Remote Debugger Stub/Preferences** (Macintosh):



   Hier müssen folgende Einstellungen vorgenommen werden:

   Machine Name: Name des Testrechners. Er wird auf der Entwicklermaschine angezeigt und dient zur Identifikation des Testrechners. Er ist frei wählbar.

   Listening Port: Port, auf dem Remote Debugger Stub TCP-Verbindungen annimmt. Der vorgegebene Port funktioniert für die meisten Anwendungsfälle. Die Option ist speziell dann interessant, wenn im Netzwerk eine Firewall verwendet wird.

   Public Debugger Stub: Falls aktiviert, kann jeder im lokalen Netzwerk feststellen, dass auf dieser Maschine ein REALbasic Debugger Stub läuft. Aktivieren Sie die Option, damit Sie auf dieser Maschine debuggen können.

   Log Access Information: Ist diese Option aktiviert, werden alle Zugriffe auf die Maschine protokolliert. Damit können Sie sehen, wann und von welcher IP-Adresse aus der Debugger benutzt. Das Protokoll wird in dem Verzeichnis abgelegt, in dem sich das Programm Remote Debugger Stub befindet.

3. Klicken Sie **OK** und sichern Sie die Einstellungen.

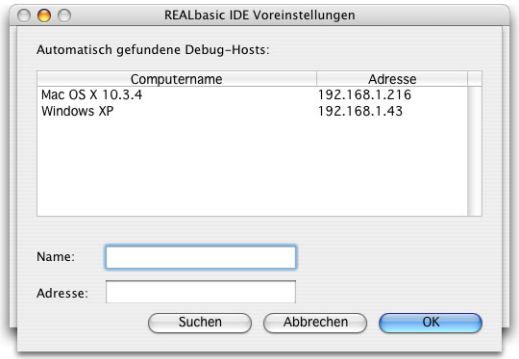
Wiederholen Sie die Schritte für jede weitere Testmaschine.

Als nächstes müssen Sie die Entwicklermaschine für Remote Debugging konfigurieren.

1. Wählen Sie den Menüpunkt **Debug/Starte Remote-Debugging/Konfiguration...** Es öffnet sich die Dialogseite „Debugger“ des REALbasic-Einstellungen-Dialogs.

Unter Remote-Debugger-Verbindungen können Sie verfügbare Testmaschinen eintragen.

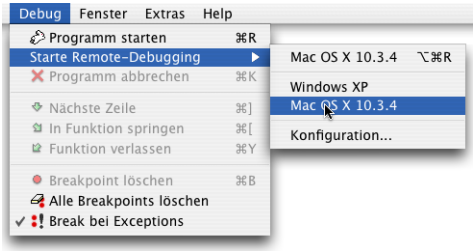
2. Klicken Sie auf „Hinzufügen“, um eine neue Testmaschine einzutragen. Es erscheint eine Liste aller Rechner im Netzwerk, auf denen zu diesem Zeitpunkt ein Remote Debugger Stub läuft.



Der Computername in der ersten Spalte entspricht jeweils dem „Machine Name“, den Sie in den Einstellungen des Remote Debugger Stubs vergeben haben.

3. Wählen Sie eine Testmaschine aus. Der Name und die IP-Adresse der Testmaschine werden in die entsprechenden Felder eingetragen. Hier können Sie auch manuell eine Testmaschine eintragen.
4. Klicken Sie **OK**. Damit übernehmen Sie den Computer in die Liste der Testmaschinen.
5. Wiederholen Sie diese Schritte für alle weiteren Testmaschinen.

Die Testmaschinen erscheinen unter **Debug/Starte Remote-Debugging** im Menü:



Die zuletzt benutzte Testmaschine erscheint ganz oben und ist auch über einen Tastatur-Shortcut erreichbar.

Um das aktuelle Projekt auf einer Testmaschine zu debuggen, gehen Sie folgendermaßen vor:

1. Vergewissern Sie sich, dass Remote Debugger Stub auf der Testmaschine gestartet wurde.
2. Wählen Sie die gewünschte Testmaschine aus dem **Starte Remote-Debugging**-Untermenü.

   Daraufhin erzeugt REALbasic eine Debug-Version Ihrer Applikation und schickt diese über das Netzwerk auf die Testmaschine, wo sie automatisch gestartet wird. Ein Fortschrittsbalken zeigt den Verlauf an.
3. Jetzt können Sie an der Testmaschine Ihre Applikation testen.

   Falls Sie Breakpoints gesetzt oder den Debugger aktiviert haben, erscheint dieser auf der Entwicklermaschine.

   Sie müssen die Debug-Version Ihres Programms beenden, bevor Sie in der Entwicklungsumgebung weiter arbeiten können. Beenden Sie dazu entweder das Programm auf der Testmaschine oder rufen Sie auf der Entwicklermaschine den Menüpunkt **Debug/Programm abbrechen auf**.

Über Firewalls

Damit Remote Debugging funktionieren kann, müssen bestimmte Ports Ihrer Firewall freigeschaltet sein. Für die UDP-Ports 44553 und 44554 sowie den TCP-Port 13897 muss ein- und ausgehender Verkehr erlaubt sein.

Den TCP-Port können Sie ändern, die UDP-Ports nicht.

Kapitel 12 Kommunikation nach draußen

Manche Anwendungen müssen mit anderen Computern kommunizieren oder auf Hardware zugreifen, die am seriellen Port angeschlossen ist.

Soll zum Beispiel eine Internet-Verbindung hergestellt werden, dann muss eine Verbindung zwischen einem Programm auf Ihrem Computer und dem Programm, das auf dem Computer Ihres Internet-Providers läuft, hergestellt werden.

REALbasic enthält Steuerelemente, die die Kommunikation von Programmen auf verschiedenen Rechnern und die Kommunikation über die serielle Schnittstelle stark vereinfachen.

Inhalt

- Kommunikation mit seriellen Geräten
- Kommunikation mit anderen Rechnern über TCP/IP
- Kommunikation mit anderen Rechnern über UDP

Kommunikation mit seriellen Geräten

Ein serielles Gerät kann serielle Datenströme empfangen oder senden, nicht beides gleichzeitig. Das gebräuchlichste serielle Gerät ist ein Modem. Auch manche Drucker werden seriell angeschlossen. Um mit solchen Geräten zu kommunizieren, verwendet man in REALbasic das Serial-Steuerelement.

Konfiguration des Serial-Steuerelements

Der erste Schritt besteht darin, ein Serial-Steuerelement auf einem Fenster abzulegen oder mittels Programmcode eine Instanz eines Serial-Steuerelements zu erzeugen. Vor der Kommunikation muss man das Serial-Steuerelement konfigurieren. Dies ist über das Eigenschaftenfenster schon während des Programmentwurfs möglich.

Einige Geräte können nur mit einer einzigen Einstellung betrieben werden – andere, wie beispielsweise Modems, können mit einer Vielzahl von Parametern benutzt werden. Hier spielt auch eine Rolle, wie das entfernte Modem, mit dem man eine Verbindung aufnimmt, konfiguriert ist. Die Voreinstellung des Serial-Steuerelements sollte mit den meisten Modems funktionieren.

Öffnen des seriellen Ports

Nach der Konfiguration des Steuerelements können Sie den Port öffnen, um die Kommunikation mit dem Gerät zu starten. Dazu rufen Sie die **Open**-Methode des Serial-Steuerelements auf. Diese Methode ist eine Funktion, die True zurückliefert, wenn die Verbindung geöffnet wurde und False, wenn etwas schiefgegangen ist. Das Beispiel zeigt, wie mit einem Steuerelement namens Serial1 ein Port geöffnet wird:

```
If Serial1.Open then
  MsgBox "Der Port wurde geöffnet."
Else
  MsgBox "Der Port wurde nicht geöffnet."
End if
```

Nachdem der Port geöffnet wurde, ist er für andere Anwendungen und sogar für andere Serial-Steuerelemente gesperrt, bis Sie ihn wieder schließen.

Daten lesen

Der DataAvailable-Event-Handler des Serial-Steuerelements wird ausgelöst, sobald Daten des seriellen Geräts empfangen werden. Die Daten werden in einem Speicherbereich abgelegt, den man Puffer (Buffer) nennt.

Im DataAvailable-Event-Handler lesen Sie die Puffer-Daten mit den Methoden Read oder ReadAll aus. Beide Methoden sind Funktionen. Mit Read lesen Sie eine bestimmte Anzahl von Bytes (Zeichen) aus dem Puffer und mit ReadAll lesen Sie den gesamten Pufferinhalt. Nach dem Lesen werden die Daten automatisch aus dem Puffer entfernt, um Platz für weitere ankommende Daten zu schaffen. Sollte es erforderlich sein, auf die Daten zuzugreifen, ohne den Pufferinhalt zu löschen, können Sie die LookAhead-Eigenschaft des Serial-Steuerelements verwenden.

So werden eingehende Daten an ein EditField angehängt:

```
Sub DataAvailable()
  EditField1.Text=EditField1.Text+Me.ReadAll()
End Sub
```

Wenn Sie die Daten im Puffer einfach nur entfernen möchten, ohne sie zu lesen, rufen Sie die **Flush**-Methode auf.

Daten schreiben

Sie können Daten zu einem seriellen Port schicken, wenn dieser erfolgreich geöffnet wurde. Dazu verwenden Sie die Write-Methode des Serial-Steuerelements. Der Write-Methode übergeben Sie eine Zeichenkette (String), die die Daten enthält.

Die Write-Methode arbeitet asynchron. Die nächste Write-Anweisung kann also bereits ausgeführt werden, bevor alle Daten des vorherigen Write tatsächlich übertragen wurden. Sollten Sie aus irgendeinem Grund warten wollen, bis die zuvor geschickten Daten versendet wurden, können Sie direkt nach dem Write die XmitWait-Methode aufrufen.

Dynamisches Ändern der Konfiguration

Manchmal muss die Konfiguration des Serial-Steuerelements verändert werden, während der Port geöffnet ist. Sie können die Eigenschaften zwar verändern, aber die Konfiguration des Ports ändert sich erst dann, wenn der Port geschlossen und erneut geöffnet wird. Sollen die Änderungen sofort wirksam werden, rufen Sie die **Poll**-Methode des Serial-Steuerelements auf. Diese Methode verändert die Einstellungen sofort und ruft den DataAvailable-Event-Handler auf, falls noch Daten im Puffer stehen, die gelesen werden müssen.

Schließen des Ports

Ist die Kommunikation mit dem seriellen Gerät beendet, müssen Sie den Port wieder schließen, um ihn für andere Anwendungen verfügbar zu machen. Dazu rufen Sie die **Close**-Methode des Serial-Steuerelements auf.

Kommunikation mit Modems

Ein Modem hat einen Befehlssatz, der z.B. einen Befehl zur Anwahl einer bestimmten Rufnummer enthält. Die meisten Modems verwenden den AT-Befehlssatz. Im Handbuch Ihres Modems sollte der verwendete Befehlssatz dokumentiert sein.

Kommunikation mit USB- und FireWire-Geräten

USB- und FireWire-Geräte benötigen spezielle Treiber. Wenn Sie ein solches Gerät ansteuern wollen, benötigen Sie eine Shared Library des Geräteherstellers oder müssen sich diese selbst schreiben.

Kommunikation über TCP/IP mit TCPSocket

Mit dem TCPSocket-Steuerelement können Sie eine TCP/IP-Verbindung innerhalb eines Netzwerks aufbauen, um mit anderen Computern zu kommunizieren.

In den Versionen vor REALbasic 5.0 wurde zur Kommunikation über TCP/IP das Socket-Steuerelement verwendet. Dieses wurde durch das TCPSocket-Steuerelement ersetzt.

Das TCPSocket-Steuerelement leitet sich von einer neuen Basis-Klasse ab, der SocketCore-Klasse. Die SocketCore-Klasse kümmert sich um die Grundfunktionen der Protokolle TCP und UDP. Die SocketCore-Klasse ist eine abstrakte Klasse, die nicht instanziiert werden kann und besitzt Eigenschaften und Methoden, die von den Steuerelementen TCPSocket, ServerSocket und UDPSocket genutzt werden. Wenn Sie das alte Socket-Steuerelement noch in Ihren Projekten verwenden, sollten Sie es durch TCPSocket ersetzen.

TCP/IP ist das Protokoll, das auch im Internet verwendet wird. TCP steht für Transmission Control Protocol, IP steht für Internet Protocol.

Wenn Sie eine Verbindung zum Internet herstellen, wird Ihr Computer Teil des Internets und kommuniziert mit anderen Computern über TCP/IP.

Konfiguration

Vor der Kommunikation mit einem TCPSocket-Steuerelement steht die Konfiguration. Die Kommunikation erfolgt auch bei TCP/IP über einen Port, nur dass es sich dabei nicht um eine Hardware-Schnittstelle Ihres Computers handelt. TCP/IP-Ports sind so etwas ähnliches wie Fernsehkanäle. Allerdings gibt es tausende Ports. Sie können sich ganz darauf konzentrieren, Daten an die Ports zu schicken und Daten aus den Ports auszulesen und müssen sich nicht mit den Details des TCP/IP-Protokolls herumschlagen. Da für E-Mail und WWW unterschiedliche Ports verwendet werden ist es möglich, gleichzeitig im Web zu browsen und E-Mails zu versenden. Jeder Port wird durch eine Nummer repräsentiert. Wenn Sie mit einer anderen Anwendung über TCP/IP kommunizieren wollen, müssen Sie wissen, welche Portnummer diese Anwendung verwendet. Ist die andere Anwendung z.B. ein SMTP-Server (Simple Mail Transfer Protocol), dann verwendet sie wahrscheinlich Port 25, da dieser für SMTP vorgesehen ist.

Ein TCPSocket-Steuerelement hat eine Port-Eigenschaft (abgeleitet von der SocketCore Klasse), die bereits während des Programmentwurfs oder später während des Programmlaufs gesetzt werden kann. Sie muss aber auf jeden Fall gesetzt sein, um eine Verbindung aufzubauen. Außerdem müssen Sie die IP-Adresse des Computers kennen, zu dem Sie eine Verbindung aufbauen wollen. Diese wird in der Address-Eigenschaft des TCPSocket-Steuerelements abgelegt.

Mac OS X und Linux schränken die freie Verwendung von Portnummern ein. Ports unterhalb von 1024 können nur von einem Benutzer mit „root"-Zugriffsrechten verwendet werden. Mac OS X ist so angelegt, dass ein Benutzer über die grafische Benutzeroberfläche keine „root"-Privilegien erlangen kann. Die meisten Anwender haben „Admin"-Zugriffsrechte. Dies ist kein Fehler, sondern eine Sicherheitsvorkehrung. Da TCPSocket nicht auf Portnummern kleiner als 1024 zugreifen kann, sollten Sie für die normale TCP/IP-Kommunikation mit Mac OS X-Computern nur Ports ab 1024 verwenden.

Anmerkung: Ein TCPSocket-Steuerelement kann nur eine Verbindung gleichzeitig herstellen. Wenn Sie mehrere Verbindungen aufbauen möchten, sollten Sie das ServerSocket-Steuerelement verwenden. Es ist speziell dafür ausgelegt. Detaillierte Informationen zu diesem Thema finden Sie im Abschnitt „Mit ServerSocket mehrere Verbindungen gleichzeitig verwalten" auf Seite 373.

Mit einem anderen Computer verbinden

Sind Port und Adresse definiert, können Sie die Verbindung zu einer Anwendung auf einem anderen Computer herstellen, sofern diese auf dem Port, den Sie angegeben haben, auf eine Verbindung wartet. Sie stellen die Verbindung mit der

Connect-Methode des TCPSocket-Steuerelements her. Es ist nicht sichergestellt, dass die Verbindung sofort zustande kommt. Aufgrund netzwerktechnischer Gegebenheiten kann es zu Verzögerungen beim Verbindungsaufbau kommen.

Es gibt zwei Möglichkeiten festzustellen, ob eine Verbindung aufgebaut wurde. Entweder Sie warten, bis der Connected-Event ausgelöst wird oder Sie prüfen den Rückgabewert der IsConnected-Methode der SocketCore-Klasse. Wenn Sie nicht auf die Bestätigung warten, kann es passieren, dass der Verbindungsaufbau abbricht und ein „Verbindung verloren"-Fehler (102) oder ein „Unbekannter Zustand"-Fehler (106) auftritt.

Ist die Verbindung aufgebaut, wird der Connected-Event-Handler ausgeführt. Konnte die Verbindung nicht aufgebaut werden oder ist ein Fehler aufgetreten, wird der Error-Event-Handler ausgeführt.

Wurde die Verbindung erfolgreich aufgebaut, kann Ihre Applikation Daten verschicken oder empfangen.

Auf eine Verbindung warten

Es gibt auch die Möglichkeit, dass Ihre Anwendung darauf warten soll, dass ein anderer Computer zu ihr Kontakt aufnimmt. Das heißt, dass Sie die Verbindung nicht aktiv aufbauen, sondern darauf warten, dass ein anderer Computer Anschluss sucht. Rufen Sie dazu die **Listen**-Methode des TCPSocket-Steuerelements auf. Wenn Sie beispielsweise ein TCPSocket-Steuerelement namens TCPSocket1 verwenden, das nach Betätigen eines Knopfes auf eine Verbindung warten soll, dann schreiben Sie folgenden Code:

```
Sub Action()
  TCPSocket1.Listen
End Sub
```

Steht die Verbindung, wird der Connected-Event-Handler des TCPSocket-Steuerelements ausgeführt.

Daten lesen

Schickt die Anwendung am anderen Ende der Verbindung Daten, wird der DataAvailable-Event-Handler des TCPSocket-Steuerelements aufgerufen. Auch hier werden die Daten zunächst in einem Puffer (Buffer) abgelegt.

Im DataAvailable-Event-Handler können Sie mit den Read- und ReadAll-Methoden des TCPSocket-Steuerelements die Daten aus dem Puffer auslesen. Mit Read lesen Sie eine bestimmte Anzahl von Bytes (Zeichen) aus dem Puffer und mit ReadAll lesen Sie den gesamten Pufferinhalt. Nach dem Lesen werden die Daten automatisch aus dem Puffer entfernt, um Platz für weitere ankommende Daten zu schaffen. Sollte es erforderlich sein, auf die Daten zuzugreifen, ohne sie aus dem Puffer zu entfernen, können Sie die LookAhead-Eigenschaft des TCPSocket-Steuerelements dazu verwenden.

So werden eingehende Daten an ein EditField angehängt:

```
Sub DataAvailable()
  EditField1.Text=EditField1.Text+Me.ReadAll()
End Sub
```

Wenn Sie Textdaten von einem anderen Computer empfangen, müssen Sie unter Umständen die Textcodierung beachten. Stammt der Text aus einer anderen Applikation, von einem anderen Betriebssystem oder ist sogar in einer anderen Sprache verfasst, müssen Sie REALbasic mitteilen, um welche Codierung es sich handelt. Weitere Informationen finden Sie im Abschnitt „Arbeiten mit Textcodierungen" auf Seite 249.

Die Methoden Read und ReadAll akzeptieren einen optionalen Parameter, um die Codierung anzugeben. Verwenden Sie das Encodings-Objekt und geben Sie die gewünschte Codierung an. Das Beispiel oben wurde entsprechend geändert, um anzugeben, dass der gelesene Text im ANSI-Format codiert ist:

```
Sub DataAvailable()
  EditField1.Text=EditField1.Text+Me.ReadAll(Encodings.WindowsANSI)
End Sub
```

Ab sofort kennt REALbasic die Codierung dieses Textes. Der Text wird also korrekt dargestellt und String-Operationen arbeiten wie erwartet.

Daten schreiben

Mit der Write-Methode des TCPSocket-Steuerelements können Sie Daten an die Anwendung senden, zu der eine Verbindung besteht. Die Write-Methode erwartet als Parameter eine Zeichenkette (String) mit den zu sendenden Daten. Im Beispiel wird der Text eines EditFields verschickt:

```
TCPSocket1.Write EditField1.Text
```

Wenn Sie den Text an eine Applikation verschicken, die eine bestimmte Codierung erwartet, müssen Sie den Text konvertieren, bevor Sie ihn verschicken. Verwenden Sie dazu die ConvertEncoding-Funktion. Diese erwartet als Parameter den Text und die Codierung. Das folgende Beispiel verschickt den Text eines EditFields in MacRoman-Codierung:

```
TCPSocket1.Write ConvertEncoding(EditField1.text,Encodings.MacRoman)
```

Wenn Sie die Write-Methode aufrufen, starten Sie einen Prozess, der die Daten auf den TCP-Socket schreibt. Je nach Implementierung und Provider gelten unterschiedliche Einschränkungen. So kann die Datenmenge, die auf einmal verschickt wird, variieren.

Davon sind nicht nur große Datenmengen betroffen. Stellen Sie also niemals Annahmen darüber an, wieviele Daten zwischen einzelnen Aufrufen der SendProgress-Methode verschickt werden. Es ist normal, dass dieser Wert schwankt.

Falls Sie mehr Daten als das vorgegebene Maximum verschicken, werden die Daten nicht auf einmal versendet. Stattdessen wird REALbasic Ihre Daten in einer Schleife verschicken und regemäßig den SendProgress-Event auslösen. Verschicken Sie zu wenige Daten, werden diese auf Netzwerkebene gepuffert. Dies bedeutet, dass die Daten vielleicht noch gar nicht verschickt wurden, obwohl Sie bereits einen SendProgress- oder SendComplete-Event erhalten haben.

Dieses Verhalten geht auf den sogenannten Nagle-Algorithmus zurück, der die Netzwerkeffizienz steigert. Jedes Datenpaket wird mit einem Header (Kopf) versehen, bevor es verschickt wird. Dies geschieht automatisch auf Netzwerkebene. Auch wenn Sie sehr oft nur ein oder zwei Bytes verschicken, werden diese jeweils mit einem 40 Bytes großen Header versehen. Ohne den Nagle-Algorithmus würde in diesem Fall das Netzwerk stark belastet werden.

Momentan erlaubt es REALbasic nicht, diese Einstellung zu deaktivieren.

Hinweis: Auch wenn Ihre Anwendung nur ein einzelnes Byte und danach nichts weiter verschickt, wird dieses natürlich übertragen. Auch mit aktiviertem Nagle-Algorithmus.

Schlussfolgerung: Gehen Sie niemals einfach davon aus, dass alle Daten versendet wurden. Warten Sie stattdessen den SendComplete-Event ab, der ausgelöst wird, sobald der Versand abgeschlossen ist. Auch der bytesSent-Parameter des SendProgress-Events kann abhängig von der übertragenen Datenmenge variieren.

Tipp: Wenn Sie speziell in einem kleinen Netzwerk nur kleine Datenpakete verschicken, sollten Sie statt TCPSocket die UDPSocket-Klasse in Betracht ziehen.

Fehlerbehandlung

Fehler können beim Verbindungsaufbau, beim Senden und beim Empfangen von Daten auftreten. Fehler sind aber nicht immer das, was sie zu sein scheinen: Die Meldungen sind mit Vorsicht zu genießen. Schließt z.B. ein anderer Computer die Verbindung, wird der Error-Event-Handler des TCPSocket-Steuerelements aufgerufen. Der Fehlercode wird in der LastErrorCode-Eigenschaft der SocketCore Klasse abgelegt. In der Sprachreferenz finden Sie unter „SocketCore“ eine Liste der Fehlercodes samt ihrer Bedeutung.

Die Fehlercodes teilen also Ihrer Anwendung mit, dass eine ordnungsgemäße Kommunikation unter den gegenwärtigen Bedingungen nicht möglich ist. Ist zum Beispiel unter Mac OS 8/9 Open Transport nicht installiert, wird ein Fehler gemeldet.

Verwaisen eines Sockets

Sockets können verwaisen. Das bedeutet, dass ein Socket auch dann weiter bestehen kann, wenn das Objekt, das ihn erzeugt hat (z.B. ein Fenster), nicht mehr existiert. Diese Eigenschaft wird für den ServerSocket benötigt. Wenn die Connect-Methode der SocketCore-Klasse oder die Listen-Methode der TCPSocket-Klasse aufgerufen oder ein Socket mit der AddSocket-Methode der ServerSocket-Klasse erzeugt wird, wird der Referenzzähler des Sockets um eins erhöht.

Beispiel: Sie haben eine eigene Sockets-Unterklasse namens „MySpiffySocket“ implementiert. Im Action-Event eines PushButtons können Sie dann folgenden Code verwenden:

```
Dim s as MySpiffySocket
s = New MySpiffySocket
s.port = 7000
s.address = "eincooler.server.de"
s.connect
```

Ein Socket bleibt also bestehen, bis er explizit geschlossen wird. Wenn Sie ein TCPSocket in ein Fenster gezogen, seine Connect- oder Listen-Methode aufgerufen und dann das Fenster geschlossen haben, gibt es zwei Referenzen auf dieses Socket. (In älteren REALbasic-Versionen gab es in diesem Fall nur eine Referenz, nämlich die des Fensters.) Der Socket wird weiterhin funktionieren, solange die Verbindung besteht, obwohl das Fenster geschlossen wurde. Wenn Sie nicht die Connect- oder Listen-Methode aufgerufen haben, existiert nur eine Referenz auf das Socket (die des Fensters), die durch Schließen des Fensters ebenfalls geschlossen wird.

Ein Socket wird geschlossen, indem Sie seine Close-Methode aufrufen oder indem die Gegenstelle die Verbindung beendet. Außerdem werden alle noch offenen Sockets beim Beenden eines Programms anstandslos geschlossen, ohne dass Speicherlecks auftreten.

Maximale Anzahl von Sockets

In Mac OS X-Versionen kleiner 10.3 können Programme nur eine begrenzte Anzahl von Sockets öffnen. Das liegt daran, dass die zugrunde liegenden BSD-Sockets für jeden Socket, der aktuell an einen Port des Rechners gebunden ist, einen Dateideskriptor belegen und Mac OS X pro Applikation nur 256 Dateideskriptoren erlaubt. Da die Anzahl der Dateideskriptoren vom verfügbaren Speicherplatz abhängig ist und ihre Applikation und die verwendeten APIs ebenfalls Dateien geöffnet haben können, stehen normalerweise weniger als 256 Sockets für Verbindungen zur Verfügung. Diese Problematik tritt unter Windows und Mac OS 9 nicht auf und ist kein Fehler von REALbasic, sondern charakteristisch für das BSD-System von Mac OS X.

Hinweis: Das Problem kann in seltenen Fällen auch unter Linux auftreten.

Schließen der Verbindung

Wenn die Kommunikation mit dem entfernten Rechner beendet ist, können Sie die Verbindung trennen, indem Sie die Close-Methode des TCPSocket-Steuerelements aufrufen.

```
TCPSocket1.Close
```

E-Mails über TCP/IP versenden und empfangen

REALbasic bringt alles mit, was Sie zum Entwickeln eines E-Mail-Programms benötigen. Ein typisches E-Mail-Programm verwendet die Protokolle POP (Post Office Protocol) und SMTP (Standard Mail Transfer Protocol). POP wird zum Empfangen, SMTP zum Verschicken von E-Mails eingesetzt.

Mit POP3Socket und SMTPSocket stellt REALbasic zwei von TCPSocket abgeleitete Klassen bereit.

- *POP3Socket* enthält Eigenschaften für Benutzernamen und Passwort und kann sich mit diesen beim E-Mail-Server anmelden, nach neuen Nachrichten suchen, Nachrichten empfangen und vom Server löschen und die Verbindung zum Server beenden.
- *SMTPSocket* arbeitet mit einer Warteschlange, in die zu versendende Nachrichten eingereiht werden. SMTPSocket bietet Methoden zum Einfügen einer Nachricht in die Warteschlange, zum Versenden der Nachrichten und zum Trennen der Serververbindung.

REALbasic stellt weitere drei Klassen zum Verwalten von E-Mails zur Verfügung:

EmailMessage enthält den Nachrichtentext, den mit *EmailHeaders* erzeugen Nachrichtenkopf und die mit *EmailAttachment* gespeicherten Anlagen. Einzelheiten zu diesen Klassen entnehmen Sie bitte der Sprachreferenz.

REALbasic Professional unterstützt sichere POP3- und SMTP-Verbindungen über die POP3SecureSocket- und SMTPSecureSocket-Klassen, die von der SSLSocket-Klasse abgeleitet wurden und ansonsten mit POP3Socket und SMTPSocket identisch sind. Indem Sie die Secure-Eigenschaft der SSLSocket-Klasse auf True setzen, sorgen Sie dafür, dass die Kommunikation verschlüsselt erfolgt.

Kommunikation über HTTP

Das HTTP-Protokoll wird von Webbrowsern eingesetzt. (HTTP steht für Hypertext Transfer Protocol.)

REALbasic stellt zur HTTP-Kommunikation die Klassen HTTPSocket und HTTPSecureSocket bereit. HTTPSocket ist eine Unterklasse von TCPSocket. HTTPSecureSocket wurde von SSLSocket abgeleitet und wird zur verschlüsselten Kommunikation benutzt (nur in REALbasic Professional verfügbar). Die Klassen enthalten Methoden zum Laden von URLs und zum Absenden von Formularen.

In REALbasic Professional steht das verschlüsselte HTTPS-Protokoll zur Verfügung. Dieses können Sie aktivieren, indem Sie die Secure-Eigenschaft der SSLSocket-Klasse auf True setzen. Dadurch ändert sich der Standard-Port von 80 auf 443.

Detaillierte Informationen zu Methoden, Events und Eigenschaften von HTTPSocket und HTTPSecureSocket finden Sie in der Sprachreferenz.

Mit ServerSocket mehrere Verbindungen gleichzeitig verwalten

Ein TCPSocket kann auf einem Port nur eine Verbindung gleichzeitig bedienen. Soll eine Anwendung mehrere Verbindungen über denselben Port unterstützen, bietet sich dafür das ServerSocket-Steuerelement an. Ein ServerSocket macht nichts anderes, als auf einem bestimmtem Port auf eingehende Verbindungen zu warten.

Geht eine Verbindung beim ServerSocket ein, übergibt es diese an einen TCPSocket und wartet auf weitere Verbindungen. Ohne ServerSocket wäre dies schwierig zu realisieren. Dazu müsste ein TCPSocket nach Eingang einer Verbindung ein neues TCPSocket erzeugen, das auf eine weitere Verbindung wartet. Dabei könnte es leicht passieren, das in der zum Erzeugen des neuen Sockets und Starten des Listen-Prozesses notwendigen Zeit eine Verbindungsanfrage eingeht, die nicht beantwortet werden kann.

Die Verbindungssuche eines ServerSockets wird gestartet, indem Sie seiner Port-Eigenschaft den abzuhörenden Port übergeben und dann seine Listen-Methode aufrufen.

Wenn Sie unter Mac OS X oder Linux versuchen, einen Port kleiner 1024 anzusprechen, erhalten Sie einen SocketCore.Error Nummer 105, es sei denn, Ihre Anwendung läuft mit root-Zugriffsrechten. Dieses Verhalten ist von den Mac OS X- und Linux-Entwicklern so gewollt und soll Sicherheitsprobleme vermeiden helfen.

Das ServerSocket-Steuerelement verwaltet einen Pool von TCPSockets, an die es eingehende Verbindungen übergibt. Mit den Eigenschaften MinimumSocketsAvailable und MaximumSocketsConnected können Sie nach dem Start des Listen-Prozesses deren Anzahl festlegen. Wenn Sie die MaximumSocketsConnected-Eigenschaft ändern, wird keine bestehende Verbindung getrennt (es werden unter Umständen lediglich keine neuen Verbindungen aufgebaut, bis die

bestehenden getrennt wurden). Das Ändern der MinimumSocketsAvailable-Eigenschaft kann den AddSocket-Event des ServerSockets auslösen, damit dieser seinen internen Puffer auffüllen kann.

Wenn Sie die Listen-Methode des ServerSockets aufrufen, bestückt dieser zunächst seinen internen Socket-Pool für eingehende Verbindungen, indem er den AddSocket-Event so oft aufruft, bis der interne Pool gefüllt ist. Der Pool umfasst die in der MinimumSocketsAvailable-Eigenschaft vorgegebene Anzahl plus zehn zusätzliche TCPSockets. (Hinweis: Wenn dieser Event Nil zurückliefert, wird dadurch eine NilObjectException ausgelöst).

Der ServerSocket kann erst dann Verbindungen übernehmen, wenn dieser Prozess abgeschlossen ist. Verbindungsanfragen, die während des Bestückens des Socket-Pools eingehen, werden abgewiesen. Wenn Sie feststellen wollen, ob der ServerSocket bereit ist, eingehende Verbindungen anzunehmen, können Sie seine IsListening-Eigenschaft abfragen.

Der AddSocket-Event eines ServerSockets kann nur einen TCPSocket oder eine Unterklasse von TCPSocket zurückliefern. Da UDP ein „verbindungsloses" Protokoll ist (siehe „Verbindungen über UDP mit UDPSocket" auf Seite 375), hätte es keinen Sinn, wenn der ServerSocket mit UDPSockets hantieren könnte.

Referenzzähler

Wenn die AddSocket-Methode einen Socket liefert, wird der Referenzzähler nicht erhöht. Statt dessen wird der Socket intern verwaltet und der Referenzzähler erst dann erhöht, wenn der ServerSocket eine Verbindung an diesen Socket übergibt. Wenn der ServerSocket vernichtet wird, bevor er Sockets aus dem internen Pool verwenden konnte, werden die unbenutzten Sockets ebenfalls vernichtet. Bis dahin ist nämlich der ServerSocket das Elternobjekt der Sockets im Pool. Im Unterschied dazu bleiben die Sockets, denen der ServerSocket eine Verbindung übertragen hat, bei Vernichtung des ServerSockets bestehen und funktionieren weiter. Das bedeutet, dass TCPSockets verwaisen können. UDPSockets und ServerSockets hingegen können nicht verwaisen, sondern müssen einen Eigentümer haben. Ist dies nicht der Fall, wird ein UDPSocket oder ServerSocket vernichtet, sobald es nicht mehr im Scope des Programms liegt. Allerdings bleiben dabei alle Verbindungen bestehen, die der ServerSocket an TCPSockets übergeben hat. Es werden nur die TCPSockets zerstört, die keine Verbindung aufgebaut haben.

Hinweis: ServerSockets stehen nur in REALbasic Professional zur Verfügung.

Sichere TCP-Verbindungen mit SSLSocket

Das SSLSocket-Steuerelement kann die Kommunikation über TCP/IP mit Hilfe einer Technik namens Secure Sockets Layer (SSL) verschlüsseln. SSL bietet eine hohe Sicherheit und wird auch zum Übertragen von Kreditkartennummern oder vertraulichen medizinischen Daten verwendet.

Aktuell unterstützt REALbasic vier verschiedene SSL-Versionen:

Tabelle 43. SSL-Protokolle in REALbasic

SSL v2	SSL Version 2
SSLv23	SSL Version 3, lässt aber auch Version 2 zu
SSLv3	SSL Version 3
TLSv11	TLS (Transport Layer Security Version 1)

Das Standardprotokoll ist SSLv23. Es ist mit den meisten Servern kompatibel. Das Protokoll muss in der Eigenschaft ConnectionType festgelegt werden, bevor mit der Connect-Methode eine Verbindung aufgebaut wird. Nach dem Aufbau einer Verbindung lässt sich das Protokoll nicht mehr ändern.

Nicht alle Server akzeptieren eine Verbindung mit dem voreingestellten Protokoll (SSLv23). Wenn Sie nicht wissen, welches Protokoll ein Server akzeptiert, probieren Sie einfach mehrere aus. Wenn der erste Versuch mit einem Fehler 102 fehlschlägt, testen Sie die nächste Version. Stellen Sie sicher, dass dieser Prozess abgebrochen wird, wenn kein Protokoll akzeptiert wird oder ein anderer Fehler auftritt.

Um eine SSL-Verbindung aufzubauen, setzen Sie die Secure-Eigenschaft auf TRUE und rufen die Connect-Methode auf.

Mit der aktuellen Implementierung von SSLSocket ist es nicht möglich, einen verschlüsselten „lauschenden" Socket zu erzeugen. Wenn Sie also die Secure-Eigenschaft auf True setzen und die Listen-Methode aufrufen, wird auf Ihrem Rechner lediglich ein unverschlüsselter Port geöffnet, der auf eine Verbindung wartet. Eine sichere Listen-Methode wird eventuell zu einem späteren Zeitpunkt in REALbasic integriert.

Für die SSLSocket- und ServerSocket-Steuerelemente gibt es keine eigenen Icons in der Werkzeugpalette. Da sich beide nicht von der Control-Klasse ableiten, können sie im Programmcode instanziiert werden. Alternativ können Sie einen SSLSocket über das Kontextmenü des Fenstereditors hinzufügen.

Hinweis: SSLSocket ist eine Unterklasse von TCPSocket und steht nur in REALbasic Professional zur Verfügung.

Verbindungen über UDP mit UDPSocket

Das User Datagram Protocol (UDP) ist die Grundlage des Hochgeschwindigkeitsnetzwerkverkehrs. UDP ist ein verbindungsloses Protokoll mit geringer Redundanz und ist nicht so sicher wie TCP. Da keine Verbindung aufgebaut werden muss, sind zur Verwendung eines UDPSocket-Steuerelements nicht annähernd so viele Schritte nötig wie beim TCPSocket-Steuerelement.

Wie TCPSocket leitet sich das UDPSocket-Steuerelement von der neuen SocketCore-Klasse ab. Anders als TCPSocket ist UDPSocket mit keinem eigenen Symbol in der Steuerelementepalette vertreten. Da UDPSocket sich aber von der SocketCore-Klasse und nicht von der Control-Klasse ableitet, ist es nicht notwendig, ein UDPSocket-Steuerelement zur Instanziierung in einem Fenster abzulegen. Sie können es per Code mit den New-Operator instanziieren.

UDPSocket muss einem bestimmten Port Ihres Computers zugewiesen werden. Sobald diese Zuweisung erfolgte, kann der UDPSocket verwendet werden. Er nimmt sofort alle am zugewiesenen Port eintreffenden Daten entgegen. Ein UDPSocket kann auch Daten verschicken.

Datagramme

Um zu unterscheiden, welche Daten von welchem Rechner kommen, verwendet UDPSocket eine Datenstruktur namens *Datagramm*. Ein Datagramm besitzt zwei Eigenschaften: *Adresse* (*Address*) und *Daten* (*Data*). *Adresse* ist die IP-Adresse des Rechners, der die Daten schickt, *Daten* sind die geschickten Daten. Auch beim Versenden von Daten müssen diese als Datagramme vorliegen. In diesen muss neben den Daten selbst die IP-Adresse des Empfängers und der Port, an den die Daten gesendet werden sollen, spezifiziert sein.

UDPSocket Modi

UDP Sockets lassen sich in verschiedenen Modi betreiben.

- Der Modus, der einer Kommunikation über TCP am ähnlichsten ist, heißt „Unicasting". Er wird verwendet, wenn Daten nur an die IP-Adresse eines einzelnen Geräts verschickt werden. Das Datagramm ist also nur für einen Empfänger bestimmt.
- Der zweite Modus nennt sich „Broadcasting" und verhält sich so, als ob Sie eine Nachricht durch ein Megaphon an alle hinausposaunen. Wenn ein Rechner den von Ihnen spezifizierten Port zufällig zu diesem Zeitpunkt abhört, empfängt er die Daten. Auf diese Weise Informationen zu versenden ist sehr netzwerklastig, da durch Broadcasting viel Datenverkehr entsteht.

  Wenn Sie Daten mittels Broadcasting versenden, verlassen die Daten Ihr Subnetz nicht, werden also nicht in das Internet hinaus geschickt.

  Um Broadcasting zu verwenden, müssen Sie die Broadcast-Adresse Ihres Rechners kennen. Diese ist von Rechner zu Rechner unterschiedlich. UDPSocket stellt die Broadcastadresse in der Eigenschaft BroadcastAddress zur Verfügung.
- Der dritte Modus, in dem sich UDPSockets betreiben lassen, heißt „Multicasting". Dabei handelt es sich um eine Kombination der Modi „Unicasting" und „Broadcasting". Multicasting verhält sich ähnlich wie ein Chatroom. Nach

dem Betreten eines Chatrooms können Sie sich mit jeder Person im Chatroom unterhalten. Wenn Sie den Chatroom betreten wollen, rufen Sie JoinMulticastGroup auf und geben an, welcher Gruppe Sie beitreten möchten. Der Group-Parameter ist eine besondere IP-Adresse, die sich *Class D IP* nennt. Sie umfasst den Bereich von 224.0.0.0 bis 239.255.255.255. Stellen Sie sich die IP-Adresse einfach als Namen des Chatrooms vor. Wenn Sie sich mit zwei anderen Leuten unterhalten möchten, müssen alle drei Teilnehmer die JoinMulticastGroup mit der selben Class D IP-Adresse als Group-Parameter aufrufen. Zum Verlassen des Chatrooms müssen Sie LeaveMulticastGroup aufrufen und wiederum die Gruppe spezifizieren, die Sie verlassen möchten. Sie können gleichzeitig an beliebig vielen Multicast-Gruppen teilnehmen. Zum Versenden von Daten an eine Multicast-Gruppe geben Sie einfach deren IP-Adresse an. Alle Rechner dieser Gruppe werden die Nachricht erhalten.

Wenn Sie die Daten, die Sie per Multicast versendet haben, auch selbst empfangen wollen, können Sie die SendToSelf-Eigenschaft (auch loopback genannt) von UDPSocket auf TRUE setzen. Sie können außerdem die Zahl der Router Hops eines Multicast-Datagramms festlegen (auch als Time to live bezeichnet). Beim Verschicken eines Datagramms durchläuft es auf dem Weg zu seinem Bestimmungsort mehrere Router. Mit jedem Router, den das Datagramm passiert, verringert sich die RouterHops-Eigenschaft um den Wert 1. Erreicht RouterHops den Wert 0, wird das Datagramm zerstört. So können Sie mit einer größeren Genauigkeit festlegen, wer Ihr Datagramm erhalten kann. Hier einige Erfahrungswerte, wie groß RouterHops gewählt werden sollte:

Tabelle 44. Empfehlungen für die RouterHops-Eigenschaft

Wert	Beschreibung
0	Gleicher Host
1	Gleiches Subnetz
32	Gleicher Ort
64	Gleiche Region
128	Gleicher Kontinent
255	ohne Einschränkung

Hinweis: Wenn Ihr Datagramm einen Router passiert, der kein Multicasting unterstützt, wird es sofort vernichtet.

Aufgrund des verbindungslosen Charakters von UDP gibt es keinerlei Garantie dafür, dass die Daten ihr Ziel erreichen. Um dieses Problem zu umgehen, können Sie ein eigenes Protokoll auf Grundlage des UDP-Protokolls entwickeln, das den Empfang von Daten bestätigt.

UDPSocket arbeitet asynchron, die Connect-Methode aber synchron. Für den Fall, dass die Verbindung nicht hergestellt werden kann, wird umgehend ein Error-Event ausgelöst und die IsConnected-Eigenschaft entsprechend gesetzt.

Netzwerke ganz einfach

Wenn Sie Netzwerkverbindungen nur zwischen REALbasic-Programmen aufbauen wollen, können Sie einige speziell dafür entwickelte Klassen verwenden. Diese basieren auf dem TCP-, beziehungsweise UDP-Protokoll, können aber nur mit anderen REALbasic-Applikationen kommunizieren. Wenn Ihr Programm beispielsweise mit einem FTP-Server kommunizieren soll, müssen Sie die allgemeineren Klassen verwenden.

Als Ausgleich für diese Einschränkung sind nur wenige Zeilen Code nötig, um eine Kommunikation aufzubauen, und es ist kein fundiertes Hintergrundwissen über die Protokolle TCP und UDP nötig.

Die AutoDiscovery Klasse

Die AutoDiscovery-Klasse organisiert die Netzwerkkommunikation zwischen REALbasic-Applikationen. Sie ist in der Lage, andere Applikationen im Netzwerk zu finden, die ebenfalls das „vereinfachte“ UDP-Protokoll als Kommunikationsgrundlage verwenden. Mit der AutoDiscovery-Klasse können Sie Nachrichten an einzelne Benutzer oder an Multicast-Gruppen schicken.

Einer Multicast-Gruppe beitreten

Um einer Multicast-Gruppe beizutreten, müssen Sie lediglich deren Namen als Parameter für die Register-Methode der AutoDiscovery Klasse angeben. Jeder, der dieser Gruppe beitreten will, verwendet denselben Namen. Die Klasse kümmert sich intern selbst um eine gültige Multicast-IP-Adresse. So treten Sie einer Multicast-Gruppe bei:

```
AutoDiscovery1.Register("MeineMulticastGruppe")
```

Um die Mitglieder einer Gruppe zu ermitteln, rufen Sie die Methode GetMemberList auf. Diese liefert ein Array von Strings mit IP-Adressen. Diese Methode können Sie immer wieder aufrufen, um die Liste zu aktualisieren.

Der folgende Code zeigt die Mitglieder einer Gruppe in einer ListBox an:

```
Dim s(-1) as string // Array deklarieren; GetMemberList vergrössert es selbst.
Dim i as Integer
s= AutoDiscovery1.GetMemberList
For i=0 to Ubound(s)  //Ubound liefert den Index des letzten Elements
  ListBox1.AddRow s(i)
Next
```

Eine Nachricht verschicken

Um eine Nachricht an eine Gruppe zu verschicken, verwenden Sie die SendMessageToGroup-Methode. Diese akzeptiert eine numerische Kommando-ID und einen Text. Das folgende Beispiel setzt den Inhalt von EditField2 als Nachricht mit der Kommando-ID 50.

```
AutoDiscovery1.SendMessageToGroup(50,EditField2.Text)
```

Um eine Nachricht an einen einzelnen Computer zu schicken, benötigen Sie dessen IP-Adresse, die Sie mithilfe der GetMemberList-Methode ermitteln können, die Kommando-ID und den Text. Die folgende Zeile schickt die gleiche Nachricht an den Computer 192.168.1.118:

```
AutoDiscovery1.SendMessageToIndividual("192.168.1.118",50,EditField2.Text)
```

Die EasyUDPSocket- und EasyTCPSocket-Klassen

Die Klassen EasyUDPSocket und EasyTCPSocket sind „vereinfachte“ Versionen der UDPSocket- und TCPSocket-Klassen. Sie sind speziell für die Kommunikation zwischen REALbasic-Applikationen ausgelegt.

Der größte Unterschied zwischen EasyTCPSocket und TCPSocket besteht darin, dass EasyTCPSocket nachrichtenbasiert arbeitet. Der Verbindungsaufbau entspricht dem des TCPSocket-Steuerelements. Zum Verschicken von Daten verwenden Sie im Unterschied zu TCPSocket die SendMessage-Methode. Geht eine Nachricht ein, die dem EasyTCPSocket-Protokoll entspricht, wird ein ReceivedMessage-Event ausgelöst.

Empfangen von Nachrichten

Das Empfangen von Nachrichten könnte einfacher nicht sein. Der ReceivedMessage-Event der AutoDiscovery-Klasse wird immer dann ausgelöst, wenn eine Nachricht von einem individuellen Rechner (inkl. Ihres eigenen) oder von einer Gruppe eingeht. Der Event-Handler liefert die IP-Adresse des Absenders, die Kommando-ID und den Nachrichtentext. Dazu reicht ein Event-Handler wie dieser völlig aus:

```
Sub ReceivedMessage(FromIP as String, Command As Integer, data as String)
  MsgBox FromIP + " sent us " + Str(Command) + ": " + data
End Sub
```

Wie Protokolle arbeiten

Jede Art der Kommunikation macht es erforderlich, dass sich die beteiligten Seiten auf eine gemeinsame Sprache einigen. So verhält es sich auch mit TCP/IP. Die verwendete Sprache nennt man in diesem Fall Protokoll. Es bestimmt die Art und Weise, wie Daten geschickt und empfangen werden.

Wenn Sie ein Programm schreiben, das eine TCP/IP-Verbindung mit einem anderen Programm herstellen soll, müssen Sie das Protokoll verstehen, das von dem anderen Programm verwendet wird. Beispielsweise wird im WWW das HTTP-Protokoll (HyperText Transfer Protocol) verwendet, für das Senden von E-Mails das SMTP-Protokoll (Simple Mail Transfer Protocol) eingesetzt und für das Empfangen von E-Mails das POP3-Protokoll (Post Office Protocol 3) benutzt. Vollständige Beschreibungen dieser und anderer Protokolle finden Sie im Internet. Man nennt diese Beschreibungen RFCs (Request For Comments). Man findet Sie zum Beispiel, indem man auf www.google.de nach „RFC“ sucht.

Wenn Sie ein Programm schreiben, das mit anderen von Ihnen selbst entwickelten Programmen kommunizieren soll, können Sie Ihr eigenes Protokoll definieren.

Kapitel 13 REALbasic erweitern

REALbasic ist unter anderem deswegen leicht erlernbar, weil es Ihnen erspart, sich mit den internen Abläufen des Betriebssystems auseinanderzusetzen. Sie müssen die vielen tausend Funktionen, die das API (Application Programming Interface) des Betriebssystems ausmachen, nicht kennen. Das bedeutet aber auch, dass es möglicherweise die eine oder andere Funktion in REALbasic (noch) nicht gibt, die Sie benötigen. Daher kann REALbasic erweitert werden.

Inhalt

- Toolbox-Funktionen aufrufen
- AppleScripts aufrufen
- Kommunikation mit AppleEvents
- Schreiben und Benutzen von REALbasic-Plugins
- Benutzung von PowerPC Shared Libraries
- Office Automation
- Active X-Komponenten

Toolbox-Funktionen aufrufen

Mit der Declare-Anweisung haben Sie Zugriff auf die Toolbox, und zwar sowohl auf der Mac- als auch auf der Windows-Plattform. Sie müssen die Declare-Aufrufe für die verschiedenen Plattformen voneinander trennen, dazu bietet sich die bedingte Compilierung an. Mit der Declare-Anweisung werden der Name der Toolbox-Funktion, die zugehörige Shared Library und die Aufrufparameter bestimmt. Falls der Aufruf einen Wert zurückliefert, müssen Sie auch den Datentyp des zurückgelieferten Wertes angeben.

Wenn eine Funktion auf beiden Plattformen zur Verfügung steht, können Sie denselben Namen für beide Plattformen verwenden. Allerdings werden die Aufrufparameter meistens unterschiedlich sein. Auch in diesem Fall bietet sich bedingte Compilierung an, um die Aufrufe für die verschiedenen Plattformen voneinander zu trennen.

Folgende Action-Methode eines Buttons verwendet den Macintosh Speech Manager, um den Text eines EditFields vorzulesen:

```
dim s as string
dim i as integer
#if TargetMacOS then
  Declare Function SpeakString lib "SpeechLib" (SpeakString as pstring) as Integer
#endif
s=editField1.text
#if TargetMacOS then
  i=SpeakString(s)
#else
  MsgBox "Sprachausgabe wird nur auf dem Macintosh unterstützt!"
#endif
```

Wenn der Name der Toolbox-Funktion mit dem Namen einer REALbasic-Methode identisch ist, müssen Sie für die Toolbox-Funktion einen Alias-Namen verwenden. Wenn z.B. SpeakString der Name einer REALbasic-Methode wäre, könnten Sie das Beispiel nicht wie oben gezeigt programmieren. Stattdessen könnten Sie folgende Syntax verwenden:

```
Declare Function MySpeakString lib "SpeechLib" Alias "SpeakString" (SpeakString as pstring) as Integer
```

Um die Toolbox-Funktion aufzurufen, müssten Sie dann MySpeakString verwenden.

Weitere Informationen und Beispiele entnehmen Sie bitte der Beschreibung des Declare-Befehls in der Sprachreferenz.

AppleScripts aufrufen

AppleScript ist die Skript-Sprache von Apple, mit der man Anwendungen auf einfache Weise steuern kann. REALbasic unterstützt AppleScript. Sie können mit dem im Mac OS enthaltenen Skripteditor ein AppleScript-Skript schreiben und es dann in Ihrem REALbasic-Projekt aufrufen.

Hinweis: AppleScript ist nur auf dem Macintosh verfügbar.

AppleScript vorbereiten

Das komplette Skript muss in einem On-Run-Handler untergebracht sein, der z.B. so aussieht:

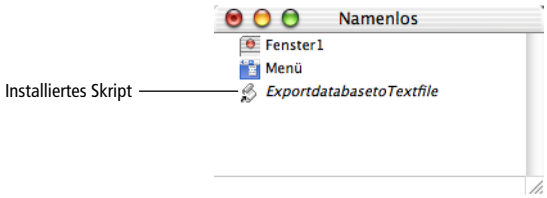
```
on run
 //Hier steht das eigentliche Skript
end run
```

Das Skript muss im Skripteditor als „Programm“ abgespeichert werden. Diese Option können Sie im „Sichern unter…“-Dialog des Skripteditors aktivieren.

Importieren eines AppleScript-Skripts

Ziehen Sie das compilierte Skript in Ihr Projektfenster. Das Skript erscheint dort mit einem AppleScript-Icon.

Abb. 275: Ein compiliertes AppleScript im Projektfenster



REALbasic speichert einen Alias des Skripts in der Projektdatei, nicht das Skript selbst. Sein Name wird kursiv dargestellt. Erst beim Erzeugen eines Stand-Alone-Programms wird das Skript in die fertige Applikation eingebaut.

Parameter an ein AppleScript übergeben

Wenn Sie ein Skript schreiben und diesem Parameter übergeben möchten, müssen Sie die Parameter hinter dem *on run*-Kommando in geschweifte Klammern setzen. Hier werden die Parameter x und y an das Skript übergeben:

```
on run {x,y}
  //Hier steht das eigentliche Skript
end run
```

Werte aus einem AppleScript-Skript zurückgeben

Skripte, die einen Wert zurückliefern, werden wie Funktionen behandelt. Verwenden Sie im Skript das *Return*-Kommando, um einen Wert zurückzuliefern. Das folgende Beispielskript erhält als Parameter das Alter in Tagen und gibt das Alter in Jahren zurück (Schaltjahre werden nicht berücksichtigt):

```
on run {daysOld}
 return daysOld/365
end run
```

Aufrufen eines AppleScript-Skripts

Skripts werden wie Methoden oder Funktionen aufgerufen.

Hier wird ein Skript aufgerufen, das die Lautstärke des Macs auf 5 setzt:

```
SetSoundLevel 5
```

Und so kann man ein Skript aufrufen, das einen Wert zurückgibt:

```
Dim level as Integer
level=GetSoundLevel()
```

Entfernen eines AppleScript-Skripts

Sie löschen ein Skript aus dem Projekt, indem Sie es im Projektfenster anklicken und dann die Backspace-Taste drücken. Alternativ dazu können Sie einen ctrl-Klick auf das Skript ausführen und im dann erscheinenden Kontextmenü den Menüpunkt **Löschen** aufrufen.

Kommunikation über AppleEvents

AppleEvents sind das Herzstück der Kommunikation zwischen den Anwendungen, die auf einem Mac laufen. Wenn Sie ein AppleScript aufrufen, dann werden alle Aktionen des Skripts über AppleEvents abgewickelt. Wenn Sie im Finder einen Neustart auslösen, schickt der Finder ein „Quit"-AppleEvent an alle laufenden Programme. Alle Programme müssen dieses AppleEvent unterstützen.

Ein AppleEvent-Objekt wird mit der Funktion NewAppleEvent erzeugt. AppleEvents bestehen aus drei Teilen: Einer Event-Klasse, einer Event-ID und dem Creator-Code der Zielanwendung.

Die Event-Klasse und die Event-ID definieren gemeinsam ein eindeutiges AppleEvent. Die Event-Klasse ist dafür gedacht, Events in Gruppen zusammenzufassen. Es gibt zahlreiche Standard-Events, von denen einige unbedingt von jedem Programm unterstützt werden müssen. Viele Programme bieten darüber hinaus auch eigene Events an, die spezielle Funktionen des Programms berücksichtigen. Informationen dazu erhalten Sie entweder in der Dokumentation des Programms oder beim Programmautor selbst.

Hinweis: AppleEvents funktionieren nur auf einem Macintosh.

AppleEvents senden

Nachdem Sie das AppleEvent-Objekt angelegt haben und die nötigen Parameter festgelegt sind, senden Sie das AppleEvent an die Zielanwendung. Dazu bedienen Sie sich der Send-Methode des AppleEvent-Objekts.

Im nächsten Beispiel wird ein AppleEvent erzeugt, das den Finder auffordert, den Mac neu zu starten. Die Klasse des Events ist „FNDR" und die Event-ID heißt „rest" (für Restart). Der Creator-Code des Finders lautet „MACS". Die Send-Methode liefert True zurück, wenn das Event erfolgreich abgeschickt werden konnte:

```
dim ae as AppleEvent
ae=newAppleEvent("FNDR","rest","MACS")
if not ae.send then
  msgBox "Der Computer konnte nicht neu gestartet werden."
end if
```

AppleEvents empfangen

Um AppleEvents empfangen zu können, muss Ihr Projekt über ein Objekt der Application-Klasse verfügen. Nur ein Application-Objekt kann auf AppleEvents reagieren, da es einen HandleAppleEvent-Event-Handler hat. Dieser empfängt das eingehende AppleEvent.

Dieser Event-Handler empfängt das AppleEvent-Objekt, die Event-Klasse und die Event-ID. Bestimmte AppleEvents muss Ihre Anwendung unterstützen. Eines davon ist das Quit-AppleEvent.

Im Beispiel wird die Quit-Methode aufgerufen, wenn der HandleAppleEvent-Event-Handler ein Quit-Event empfängt:

```
Function HandleAppleEvent(Event as AppleEvent, eventClass as String, EventID as String) as Boolean
  if eventClass="aevt" and eventID="quit" then
    //Der Finder fordert das Programm zur Beendigung auf
    beep
    msgBox "Jetzt ist leider Schluss."
    quit
  end if
End Function
```

Sie können für Ihr Programm eigene AppleEvent-Klassen und Event-IDs definieren, auf die Ihr Programm mit verschiedenen Aktionen reagieren kann.

Wenn Ihr Programm feststellen soll, ob der Mac gerade aus dem Ruhezustand erwacht ist, müssen Sie nur auf den „wake"-AppleEvent reagieren, der dann vom Mac OS an alle Applikationen verschickt wird.

Dazu muss es in Ihrem Projekt eine Application-Klasse geben. Der HandleAppleEvent-Event-Handler der Application-Klasse wird jedes Mal ausgelöst, wenn Ihre Applikation einen AppleEvent erhält. Dem Event-Handler wird dabei der AppleEvent, die EventClass und die EventID übergeben. Um festzustellen, ob es sich um die Aufwach-Meldung handelt, überprüfen Sie, ob EventClass gleich „pmgt" und eventID gleich „wake" sind. Wenn der AppleEvent diese Werte enthält, ist der Mac gerade aus dem Ruhezustand erwacht.

Hinweis: AppleEvent-Klassen und IDs sind case-sensitiv. AppleEvents können nicht empfangen werden, wenn ein Programm im Debugger der IDE abläuft. Nur die Stand-Alone-Applikation kann AppleEvents empfangen.

AppleEvents für Fortgeschrittene

AppleEvents können sehr spezialisierte Daten enthalten. Es ist möglich, AppleEvents zu programmieren, die Daten für einen Web-Server verarbeiten. Mehr Informationen finden Sie unter AppleEvent-Klasse in der Sprachreferenz.

Schreiben und Benutzen von REALbasic Plugins

Viele Anwendungen verfügen über ein eigenes Plugin-Format. Netscape Navigator, Adobe PhotoShop oder 4th Dimension sind nur einige Beispiele. Plugins erweitern das Leistungsspektrum eines Programms durch externe Module. Zum Beispiel gibt es ein Plugin für den Navigator, mit dem man QuickTime-Filme ansehen kann, die in Web-Seiten eingebunden sind.

REALbasic hat ebenfalls ein eigenes Plugin-Format. Plugins werden in C oder C++ geschrieben. James Milne von Essence Software hat zum Beispiel ein REALbasic-Plugin geschrieben, das bestimmte Sounddateien abspielen kann. Auch für die Anbindung an Datenbankserver verwendet REALbasic Plugins. Sie können auch andere Datenbank-Engines als die standardmäßig unterstützten verwenden, indem Sie ein entsprechendes Plugin programmieren oder von einem Dritthersteller erwerben.

Wenn Ihre Applikation für Mac OS X gedacht ist, müssen die Plugins carbonisiert sein.

Plugins installieren

Plugins werden einfach in einem Ordner namens „Plugins" abgelegt. Dieser muss sich im selben Verzeichnis befinden, in dem auch das REALbasic-Programm liegt. Dort findet REALbasic sie automatisch.

Plugins benutzen

Einige Plugins erscheinen als Steuerelemente in der Steuerelementepalette von REALbasic, und zwar unterhalb der eingebauten Steuerelemente.

Ein Plugin-Steuerelement wird genauso wie jedes andere Element der Steuerelementepalette verwendet, indem Sie es auf ein Fenster ziehen. Das Eigenschaftenfenster zeigt dann die Eigenschaften des Plugin-Steuerelements an.

Plugins können aber auch aus einer Sammlung von Methoden bestehen, die kein Benutzer-Interface haben. Sie erscheinen dann auch nicht im Benutzer-Interface von REALbasic. Diese Plugins müssen über eine Dokumentation verfügen, die beschreibt, welche Methoden das Plugin bietet und wie man sie einsetzt.

Plugins im fertigen Programm

Beim Erzeugen einer Stand-Alone-Anwendung werden alle Plugins automatisch im Programm abgelegt.

Eigene Plugins schreiben

Wenn Sie C oder C++ beherrschen, können Sie selbst Plugins für REALbasic schreiben. Das REALbasic Plugin Software Development Kit (SDK) ist auf der REALbasic CD und auf dem FTP-Server von REAL Software zu finden und enthält alle benötigten Informationen und Beispieldateien für den Metrowerks CodeWarrior.

Benutzung von PowerPC Shared Libraries

PowerPC Shared Libraries sind Dateien mit Routinen, die Sie aufrufen können und denen auch Parameter übergeben werden können. Wie der Name schon andeutet, laufen die PowerPC Shared Libraries nur auf Macs mit PowerPC-Prozessor. Außerdem müssen sie carbonisiert sein, wenn sie unter Mac OS X verwendet werden sollen. Shared Libraries können ein guter Weg sein, um extern Code unterzubringen, was insbesondere dann sinnvoll ist, wenn sich mehrere Programme denselben Code teilen sollen.

Zugriff auf Kommandos in Shared Libraries

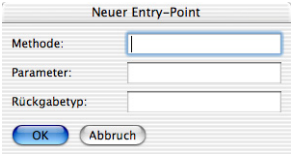
Um Kommandos (oder auch „Entry Points“) in Shared Libraries aufzurufen, müssen Sie die Shared Library in Ihr Projektfenster ziehen.

Danach müssen Sie die Entry-Points definieren, auf die Sie zugreifen möchten. Dazu verwenden Sie den Entry-Point-Editor. Sie können diesen aufrufen, indem Sie im Projektfenster einen Doppelklick auf die Shared Library ausführen. Im Entry-Point-Editor werden die von Ihnen definierten Entry-Points angezeigt.

Abb. 276: Der Entry-Point-Editor

Klicken Sie den Knopf **Neu…**, um einen neuen Entry-Point zu definieren oder selektieren Sie einen vorhandenen Entry-Point und klicken auf **Ändern**, um diesen zu verändern. Im darauf folgenden Fenster geben Sie die Parameter ein.

Abb. 277: Das Neuer Entry-Point-Fenster



Sie müssen die Namen der Entry-Points und die Datentypen ihrer Parameter kennen, um die Library verwenden zu können (siehe Dokumentation der jeweiligen Library).

Kommandos in Shared Libraries aufrufen

Einen Entry-Point einer Shared Library rufen Sie so auf, als ob es sich um eine normale Methode handeln würde. Im Beispiel wird der HMSetBalloons-Entry-Point aufgerufen, mit dem die Balloon-Help ein- und ausgeschaltet werden kann. Da dies eine Funktion ist, wird das Ergebnis der Variablen `err` zugewiesen:

```
Dim err as Integer
err = InterfaceLib.HMSetBalloons(1)
```

Microsoft Office Automation

REALbasic enthält vier Klassen, mit denen Sie Microsoft Office Applikationen direkt von REALbasic aus programmieren können. Auf dem Macintosh wird Office X, unter Windows Office 2000 unterstützt. Unter Mac OS 8/9 ist Office Automation nicht verfügbar. Natürlich muss auch Microsoft Office auf dem Rechner installiert sein, auf dem die erzeugte Applikation gestartet werden wird.

Tabelle 45. Die vier Klassen der Office Automation

Name	Beschreibung
ExcelApplication	Abgeleitet vom OLE-Objekt und automatisiert Excel
PowerPointApplication	Abgeleitet vom OLE-Objekt und automatisiert PowerPoint
WordApplication	Abgeleitet vom OLE-Objekt und automatisiert Word
Office	Enthält alle weiteren Bestandteile zur Automatisierung der Office Applikationen

Hinweise zur Office Automation unter Mac OS X

Da die OLE-Bibliotheken privat im Office-Paket gespeichert sind, müssen Sie Ihre compilierte Anwendung in den Office-Ordner innerhalb des Office X-Ordners legen (normalerweise /Programme/Microsoft Office X/Office). Damit Sie Ihre Office-Automation-Anwendung debuggen können, müssen Sie sie ebenfalls in diesen Ordner legen. In einer zukünftigen Version soll es möglich sein, die Anwendungen auch außerhalb des Office-Ordners zu benutzen.

Office Automation under REALbasic im Vergleich zu VisualBasic

Ausgehend von der Application-Klasse: Wenn Sie VBA-Code innerhalb Excel, PowerPoint oder Word schreiben, gibt es bereits eine Instanz der Application-Klasse. Beispiel:

```
Dim pres as Presentation
Dim slide1 as Slide
Set pres = Presentations.Add
' The above is the same as saying:
Set pres = Application.Presentations.Add
Set slide1 = pres.Slides.Add(1, ppLayoutText)
```

In PowerPoint funktioniert dieser Code, da bekannt ist, was ein Presentation-Objekt ist. Wenn Sie diesen Code in Excel oder Word eingeben, wird dies zu einem Fehler führen. Wie würde dieser Code in RB aussehen? Hier ist der Code:

```
Dim PowerPoint as new PowerPointApplication
Dim pres as PowerPointPresentation
Dim slide1 as PowerPointSlide
pres = PowerPoint.Presentations.Add
slide1 = pres.Slides.Add(1, Office.ppLayoutText)
```

Es gibt hier tatsächlich nur zwei Unterschiede. Der erste ist, dass alle PowerPoint-Objekte mit „PowerPoint" beginnen, alle Word-Objekte mit „Word" und alle Excel-Objekte mit „Excel". Der zweite Unterschied ist, dass sich alle konstanten Werte im Office-Modul befinden. Damit gibt es keine Probleme im Namensraum in REALbasic.

Übergeben von benannten Parametern: Wird von REALbasic nicht unterstützt. Mit etwas Aufwand können Sie es trotzdem realsieren. Zunächst müssen Sie sich mit dem OLEObject vertraut machen. Dazu können Sie in der Sprachreferenz nachschlagen oder folgendes Beispiel als Grundlage nehmen. Im Beispiel zeichnen wir mit Word ein „Suchen und Ersetzen"-Makro auf und realisieren dann dasselbe mit REALbasic.

Das Word-Makro:

```
Selection.Find.ClearFormatting
Selection.Find.Replacement.ClearnFormatting
With Selection.Find
.Text = "find this"
.Replacement.Text = "replace with"
.Wrap = wdFindContinue
.Format = false
.MatchCase = false
.MatchWholeWord = false
.MatchWildcards = false
.MatchSoundsLike = false
.MatchAllWordForms = false
End With
Selection.Find.Execute Replace:=wdReplaceAll
```

Der passende REALbasic-Code:

```
Dim word as new WordApplication
Dim find as WordFind
find = word.Selection.Find
find.ClearFormatting
find.Replacement.ClearFormatting
find.text = "find this"
find.Replacement.Text = "replace with"
find.Wrap = Office.wdFindContinue
find.Format = false
find.MatchCase = false
find.MatchWholeWord = false
find.MatchWildcards = false
find.MatchSoundsLike = false
find.MatchAllWordForms = false
// Now the fun stuff
Dim replaceParam as new OLEParameter
replaceParam.Value = Office.wdReplaceAll
// according to the docs on Find.Execute the Replace parameter is the 11th
replaceParam.Position = 11
find.Execute replaceParam
```

Das war schon alles. Der schwierigste Teil ist es, die Position des benannten Parameters herauszufinden. Dazu müssen Sie VBA starten und die Position im VBA-Objekt-Browser ermitteln.

Konflikte mit Schlüsselwörtern: Es gibt bestimmte Wörter in RB (diese werden normalerweise hervorgehoben dargestellt, wie „Select" oder „End"), die nicht als Namen von Methoden oder Eigenschaften verwendet werden können. Unglücklicherweise verwendet Excel einige dieser Namen. Um dieses Problem zu umgehen, können Sie der Eigenschaft oder Methode einen Unterstrich anhängen. Zum Beispiel:

```
Excel.Range("A1", "A3").Select_
```

Da „Select" ein Schlüsselwort ist, können Sie in einem solchen Fall den Unterstrich anhängen und RB wird dies als „Select" an Excel weitergeben.

Laden der Office Dokumentation: Wenn Ihnen die Auto-Vervollständigung nicht ausreicht oder Sie weitere Informationen über die Office-Klassen benötigen, können Sie VBA starten und über den Objekt-Browser die Klassen näher betrachten. Unter Office für Windows können Sie in den Office-Type-Libraries nachschlagen, die die Dokumentation für PowerPoint, Word und Excel enthalten. Um diese zu laden, wählen Sie den Menüpunkt **Datei/ActiveX Komponente hinzufügen…** Unter dem Karteireiter „Referenzen" können Sie die entsprechende Library auswählen. Beispielsweise enthält die „Microsoft Word #.# Object Library" die Dokumentation für Word, wobei #.# die Versionsnummer ist.

Problembehebung

Wie finde ich heraus, ob die OLE-Bibliotheken installiert sind?

Eine Möglichkeit ist es, in Word unter **Extras/Makro** den **Visual Basic-Editor** zu starten und die Automation anzuwenden.

Folgende Schritte sind dazu nötig:

1. Starten Sie Word.
2. Öffnen Sie den Visual Basic Editor.
3. Fügen Sie ein Benutzerformular ein.
4. Legen Sie einen Knopf auf das Formular.
5. Schreiben Sie folgenden Code in den Click-Event des Knopfes:
   ```
   Dim obj as Object
   Set obj = CreateObject("PowerPoint.Application")
   obj.Activate
   ```
6. Starten Sie das Programm und klicken den Knopf an.
7. Wenn PowerPoint startet und Sie keine Fehler erhalten, sind die OLE Bibliotheken installiert.

Wie kann ich auf Fehler reagieren?

Da Fehler von OLE durchgereicht werden, müssen Sie OLE-Exceptions abfangen. Diese liefern das letzte Kommando, das fehlschlug, zusammen mit weiteren Informationen über die Exception.

```
Dim word as new WordApplication
word.ShowClipboard
exception err as OLEException
msgbox err.message
```

Weiterführende Dokumentation

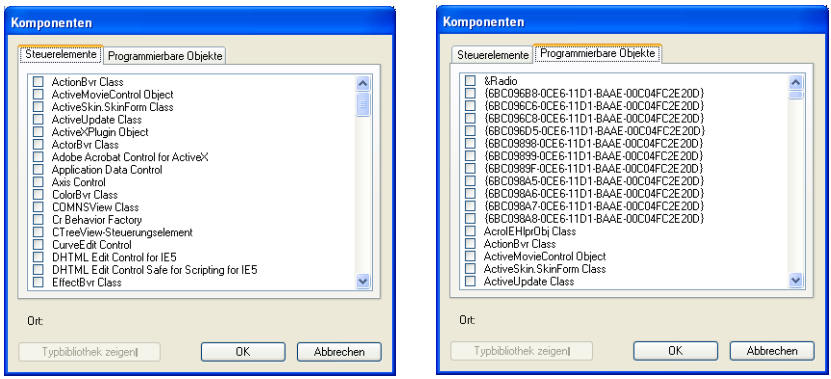
- Microsoft Visual Basic for Applications (VBA) Online-Hilfe
- www.microsoft.com/office/developer/default.html

- „Developing Solutions with Office 2000 Components and VBA“ von Peter G. Aitken
- Microsoft Office 2000/Visual Basic Programmer’s Guide

ActiveX-Komponenten

ActiveX-Komponenten sind standardisierte User-Interface-Elemente oder programmierbare Objekte, mit deren Hilfe Sie sehr schnell eine spezialisierte Benutzeroberfläche entwickeln können. In der Windows-Version von REALbasic fügen Sie der Steuerelementepalette ActiveX-Steuerelemente und -Komponenten hinzu, indem Sie den Menübefehl **Datei/ ActiveX Komponenten hinzufügen** aufrufen. Dann erscheint eine Liste der auf Ihrem PC verfügbaren ActiveX-Steuerelemente und -Komponenten.

Abb. 278: ActiveX-Komponenten



Wenn Sie ActiveX-Steuerelemente oder -Komponenten auswählen, werden diese der Steuerelementepalette abgelegt. Danach können Sie ActiveX-Steuerelemente und -Komponenten wie reguläre REALbasic-Objekte verwenden.

Detaillierte Informationen über die Programmierung von ActiveX-Komponenten finden Sie bei Microsoft in der MSDN Bibliothek unter:

`http://msdn.microsoft.com/library/`

Kapitel 14 Stand-Alone-Programme erzeugen

Beim Erzeugen einer eigenständig ablauffähigen Anwendung („Stand-Alone-Applikation“) aus Ihrem Projekt sollten Sie einige Dinge beachten, die in diesem Kapitel vermittelt werden.

Mit REALbasic können Sie zwei Arten von Stand-Alone-Applikationen erzeugen: Vollversionen und Demos. Die Vollversion einer Stand-Alone-Applikation verhält sich genauso wie eine Applikation, die Sie aus der Entwicklungsumgebung über **Debug/Run** starten, mit dem Unterschied, dass dafür kein REALbasic erforderlich ist (deswegen auch Stand-Alone-Applikation). Eine Demoversion hat die Funktionalität der Vollversion mit der Einschränkung, dass sie sich nach fünf Minuten automatisch beendet.

Mit der Professional-Version von REALbasic lassen sich Stand-Alone-Applikationen als Vollversion für Macintosh, Windows und Linux erzeugen. Mit der Standard-Version von REALbasic können Sie nur für diejenige Plattform Vollversionen erstellen, unter der Ihr REALbasic läuft. Für die anderen Plattform können Sie lediglich Demversionen erstellen.

Inhalt

- Das Programm erzeugen
- Bestandteile des Projektfensters
- Eigene Icons verwenden
- Den Creator-Code registrieren
- Verwenden und Programmieren von REALbasic-Plugins
- Der Thread-Manager

Das Programm erzeugen

Ein Stand-Alone-Applikation wird erzeugt, indem Sie **Ablage/Applikation erzeugen** aufrufen oder Shift-⌘-M (Windows: Strg-Shift-M) drücken.

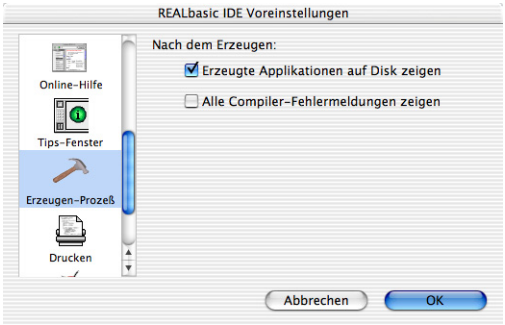
REALbasic wird eine Stand-Alone-Applikation für das Betriebssytems erzeugen, das auf Ihrem Rechner läuft (Mac OS X, Classic oder Windows). Standardmäßig erhält Ihre Applikation den Namen „Mein Programm“ oder „Mein Programm (Mac OS X)“ und wird im selben Verzeichnis wie Ihr Projekt abgelegt.

Jede Stand-Alone-Applikation enthält eine Bibliothek nahezu aller Steuerelemente, Klassen, globalen Methoden etc., die REALbasic unterstützt. Dazu gehören auch alle Plug-ins, die in Ihrer Anwendung Verwendung finden. Dies bedeutet auch, dass Ihre Stand-Alone-Applikation, zum Beispiel Code für den Umgang mit Listboxen enthält, auch wenn Sie in Ihrer Anwendung von ListBoxen keinen Gebrauch machen.

Sobald REALbasic die Applikation erzeugt hat, wird das Schreibtischfenster mit der Applikation in den Vordergrund geholt. Sie können dann mit einem Doppelklick die Anwendung testen.

Wenn Sie nicht möchten, dass das Schreibtischfenster mit der Applikation in den Vordergrund rückt, können Sie diese Einstellung unter **REALbasic/Einstellungen/Erzeugen-Prozess** (**Bearbeiten/Einstellungen** unter Mac OS Classic und Windows) deaktivieren.

Abb. 279: Einstellungen für den Erzeugen-Prozess



Beim Erzeugen einer Applikation wird eine eventuell vorhandene Datei gleichen Namens ohne Rückfrage überschrieben. Wenn Sie eine ältere Version Ihrer Anwendung behalten möchten, müssen Sie diese in einen anderen Ordner verschieben, bevor Sie eine neue Version der Applikation erzeugen.

Eine Applikation automatisch erzeugen

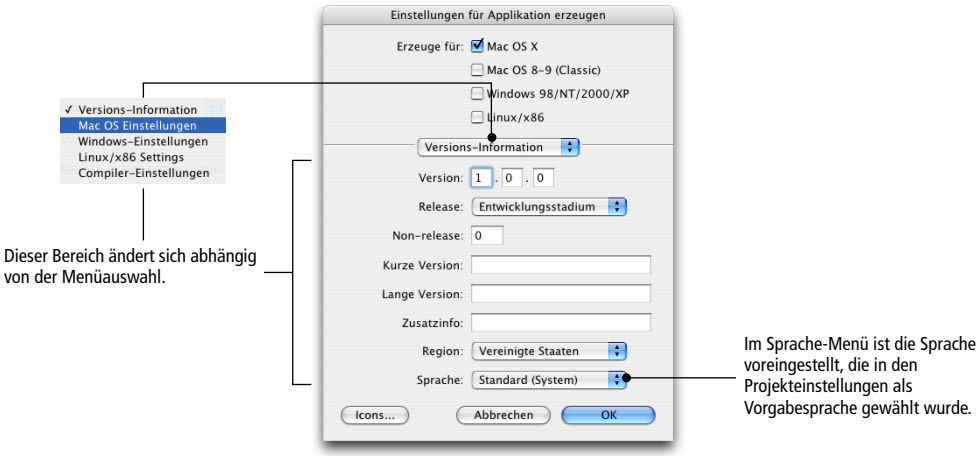
Sie können eine Applikation auch über ein AppleScript erzeugen. Das REALbasic-Dictionary enthält dazu den Build-Befehl. Der Build-Befehl verweist auf das REALbasic-Projekt. Wird er ausgeführt, erzeugt er gemäß den Compiler-Einstellungen eine oder mehrere Stand-Alone-Applikationen.

Eigenschaften der Stand-Alone-Applikation festlegen

Alle Einstellungen, die mit dem Erzeugen einer Stand-Alone-Anwendung zusammen hängen, finden Sie im Dialog **Ablage/Compiler-Einstellungen**. Hier können Sie der Applikation einen eigenen Namen und eigene Programm-Icons geben und festlegen, für welche Zielplattformen Programme erzeugt werden sollen.

1. Wählen Sie **Ablage/Compiler-Einstellungen** oder drücken Sie alt-Shift-⌘-M.

Abb. 280: Die „Einstellungen für Applikation erzeugen"-Dialogbox



Der Dialog verwendet drei Auswahlebenen, die Sie über das Popup-Menü ansteuern können.

Die Checkboxen hinter „Erzeuge für:“ legen die Zielplattform für Ihre Applikation fest. Sie können für mehrere Plattformen gleichzeitig eine Applikation erzeugen. Per Voreinstellung erzeugt REALbasic nur für das Betriebssystem eine Applikation, unter dem Sie arbeiten. Sie haben folgende Auswahl:

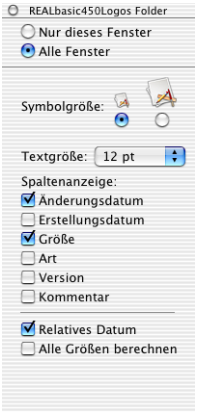
- Mac OS X: Erzeugt eine Applikation im Macho-O oder PEF-Format.
- Mac OS 8-9: Erzeugt eine Applikation für Mac OS 8/9 (PPC Version). 68k-Programme können nicht erzeugt werden. Die letzte Version, die dies unterstützt, ist REALbasic 3.5.2.
- Windows 98, NT, 2000, XP: Erzeugt eine Applikation, die unter den genannten Windows-Versionen lauffähig ist.
- Linux/x86: Erzeugt eine Applikation für x86-Linux. Linux-Applikationen benötigen GTK+ 2.0 oder höher. Für weitere Informationen besuchen Sie die Webseiten unter http://www.gtk.org.

Versions-Information

Die meisten Informationen, die Sie in der Rubrik **Versions-Information** eingeben können, erscheinen, wenn ein Mac-Benutzer im Finder auf das Programmsymbol klickt und **Ablage/Information/Allgemeine Information** (⌘-I) aufruft. Der Text, den Sie unter **Zusatzinfo** eintragen können, erscheint direkt unterhalb des Programmnamens. Der Text aus dem Feld **Lange Version** erscheint unterhalb des Änderungsdatums.

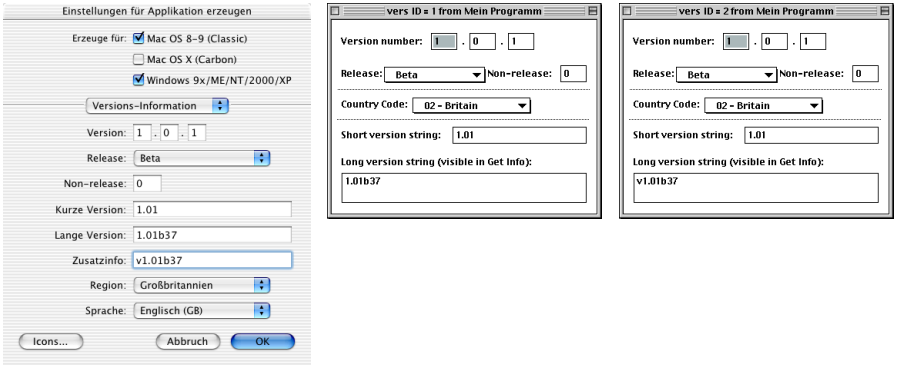
Wenn der Benutzer im Finder **Bearbeiten/Voreinstellungen** wählt und unter **Darstellungen** die **Version**-Checkbox der Spaltenanzeige aktiviert, wird in der Listenansicht des Finders die **Kurze Version** angezeigt.

Unter Mac OS X finden Sie diese Option für Fenster in Listenansicht unter **Darstellung/Darstellungsoptionen einblenden**:



Alle anderen **Informationen** sind nicht sichtbar, werden aber in der **vers**-Resource Ihrer Anwendung gespeichert. In der **vers**-Resource lassen sich Informationen zu einer einzelnen Anwendung oder einer Gruppe von Dateien ablegen. Eine **vers**-Resource mit ID=1 steht für die Versions-Information einer Applikation. Eine **vers**-Resource mit ID=2 steht für die Versions-Informationen einer Gruppe von Dateien. Die folgende Abbildung veranschaulicht die Beziehung zwischen den **Versions-Information**-Einstellungen und den Einstellungen für die **vers**-Resource.

Abb. 281: Compiler-Einstellungen in REALbasic und „vers"-Resources in ResEdit



Auch wenn die von ResEdit verwendete Namensgebung etwas verwirren mag, entspricht der String im **Long version**-Feld für **vers ID2** dem **Zusatzinfo**-Feld der **Compiler-Einstellungen**-Dialogbox in REALbasic.

Folgende Elemente werden gespeichert:

- Hauptversionsnummer im binär-codierten Dezimalformat. Zugriff über die **vers**-Resource im ersten Byte.
- Unterversionsnummer im binär-codierten Dezimalformat. Zugriff über die **vers**-Resource im zweiten Byte.
- Das Release-Stadium. Zugriff über die **vers**-Resource im dritten Byte.

  Die Codierung der Entwicklungsstadien einer Release ist wie folgt:

  Tabelle 46. Werte der Release-Stadien

Wert	Beschreibung
x20	Entwicklung
x40	Alpha
x60	Beta
x80	Release

- Pre-Release-Stadium, Zugriff über die **vers**-Resource.
- Region: Kennzeichnet das Schreibssystem, für das die Applikation gedacht ist. Die in der **vers**-Resource aufbewahrten Werte werden in der Tabelle 287 auf Seite 401 aufgelistet.
- Kurze Version: Gibt Auskunft über die Version der Software. Es obliegt dem Endbenutzer, ob er sich diese Information in der Listenansicht des Finders anzeigen lassen möchte.
- Lange Version/Paketinfo: Enthält die Versionsnummer und weitere Informationen (Firma, Entwickler, Paket). Für Dateien mit **vers ID1** wird die lange Version im Versions-Abschnitt des **Informationen**-Fensters im Finders angezeigt. Für Dateien mit **vers ID2** wird der String des **Zusatzinfo**-Feldes im **Informationen**-Fenster angezeigt.

  Weitere Informationen zur **vers**-Resource finden Sie unter:

  http://developer.apple.com/techpubs/mac/Toolbox/Toolbox-454.html

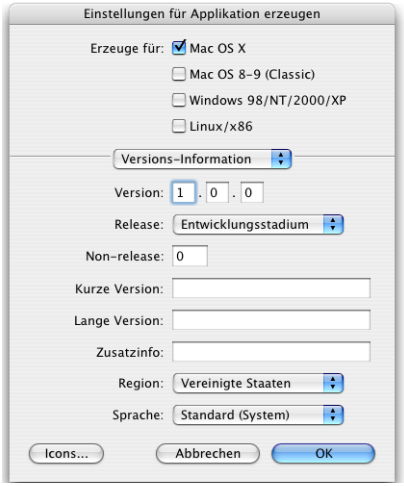
  Über die GetResource-Methode der ResourceFork-Klasse können Sie auf die in der **vers**-Resource gespeicherten Informationen zugreifen.

Mac OS-Einstellungen

Unter **Mac OS-Einstellungen** können Sie:

- einen Namen für Ihre Mac OS 8-9- oder Mac OS X-Applikation festlegen,
- für Mac OS 8/9-Applikationen die minimale und bevorzugte Speichergröße angeben,

- zwischen PEF- und Mach-O Format für Mac OS X wählen,
- einen Creator-Code für Ihre Anwendung festlegen und diesen registrieren. Der Creator-Code kann auch über eine globale Konstante gesetzt werden, sofern diese in einem Modul deklariert oder öffentlich zugänglich ist. Dazu verwenden Sie die Syntax *modulname.konstantenname* oder *fenstername.konstantenname*.



Creator-Code registrieren

Der Creator-Code jeder Applikation sollte einzigartig sein, da der Finder über den Creator-Code bestimmt, welches Programm bei einem Doppelklick auf ein Dokument geöffnet wird. Der Finder macht die erste Applikation ausfindig, deren Creator-Code mit dem Creator-Code des Dokuments übereinstimmt.

Um sicherzustellen, dass der Creator-Code Ihrer Applikation einzigartig ist, können Sie ihn bei Apple registrieren lassen. Mit Klick auf den **Registrieren**-Knopf öffnet REALbasic Ihren Webbrowser und ruft die FAQ-Seite zum Thema Registrieren von Creator-Codes auf. Die Adresse lautet:

http://developer.apple.com/dev/cftype/faq.html

Speichereinstellungen

Für Mac OS Classic-Programme müssen Sie den minimalen und den bevorzugten Speicherbedarf festlegen. Mac OS X verwaltet den Speicher dynamisch, deshalb sind diese Einstellungen nicht für Mac OS X-Programme relevant.

Per Voreinstellung trägt REALbasic 4096K als bevorzugte und 2048K als minimale Speicherzuteilung ein. Ist das zu wenig oder zu viel? Wie kann man abschätzen, wieviel Speicher eine Applikation benötigt? Ganz einfach lässt sich das leider nicht sagen. Hinweise zu den Speicheranforderungen Ihrer Applikation erhalten Sie beispielsweise über den Befehl **Über diesen Computer** im Apfel-Menü des Finders (Mac OS 8/9). Sie erhalten eine Liste mit allen aktiven Programmen, dem Speicherplatz, der für jede Applikation reserviert ist und dem tatsächlich benötigten Speicherplatz.

Das Laden von Daten, insbesondere Bildern, erhöht den Speicherbedarf. Wenn Sie die Sprite-Engine verwenden, benötigen Sie um so mehr Speicher, je mehr Sprites Sie gleichzeitig darstellen möchten. Leider gibt es keine zwingende Logik, anhand derer Sie die Speicheranforderungen Ihrer Applikation bestimmen können. Sie müssen experimentieren. Führen Sie Ihre Applikation mit verschiedenen Speichereinstellungen aus, bis Sie eine Einstellung für die minimale Speichergröße gefunden haben, unter der Ihre Applikation korrekt läuft. Anschließend korrigieren Sie Ihre Einstellungen in der **Compiler-Einstellungen**-Dialogbox und überprüfen das Resultat.

Je größer das Projekt, desto mehr Speicher benötigt die erzeugte Applikation. Wenn REALbasic eine Stand-Alone-Applikation nicht erfolgreich erzeugen kann, müssen Sie REALbasic mehr Speicher zuweisen (nur unter Mac OS 8/9).

Kompatibilitätseinstellungen

Kompatibilitätseinstellungen betreffen nur Mac OS X-Applikationen. REALbasic kann Mac OS X-Applikationen in zwei Formaten erzeugen: PEF und Mach-O. PEF-Applikationen können auch unter Mac OS 8-9 gestartet werden, sofern die CarbonLib installiert ist. Mach-O-Applikationen laufen nur unter Mac OS X.

Die PEF-Struktur speichert Programme im Format der Code-Fragment-Manager-basierten Laufzeitarchitektur. Mach hingegen ist das Format des Unix-Kernels, der Mac OS X zugrunde liegt und Mach-O sind entsprechende Objekt-Dateien. Mach-O ist das native Mac OS X-Format und wird von Apple bevorzugt. Kommandozeilen-Applikationen können nur im Mach-O-Format, Desktop-Applikationen in beiden Formaten erzeugt werden.

Windows-Einstellungen

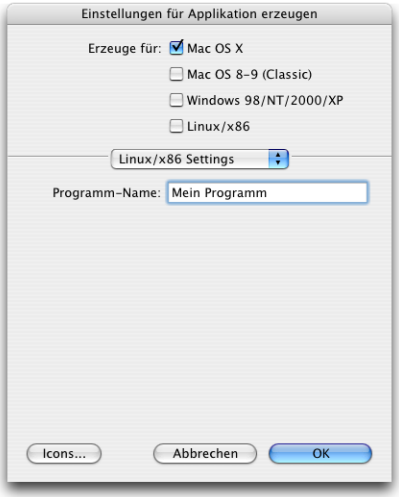
Unter **Windows-Einstellungen** legen Sie den Programmnamen der Windows-Version Ihrer Applikation fest. Sie können bestimmen, ob Ihre Applikation als Multiple Document Interface-Applikation (MDI) laufen soll. Die entsprechende Checkbox legt fest, dass die Applikation in einem MDI-Fenster ausgeführt wird. Wenn Sie sich für diese Option entscheiden, können Sie den Titel des MDI-Elternfensters vorgeben. MDI bedeutet, dass alle Fenster Ihrer Applikation in einem übergeordneten Fenster (Elternfenster) erscheinen.

Wenn Sie **Multiple Document Interface** nicht anwählen, erscheinen die Fenster Ihrer Applikation neben den Fenstern anderer offener Windows-Anwendungen und haben eigene Menüs. Wenn Ihre Applikation mehr als ein Fenster aufrufen kann, startet sie sich dazu einfach ein weiteres Mal.

Mit Hilfe der MDIWindow-Klasse können Sie diverse Eigenschaften des übergeordneten MDI-Elternfensters bestimmen, so zum Beispiel Ausgangsgröße und Ausgangsposition, Mindestgröße und Titel. Weitere Informationen zur MDIWindow-Klasse finden Sie in der Sprachreferenz.

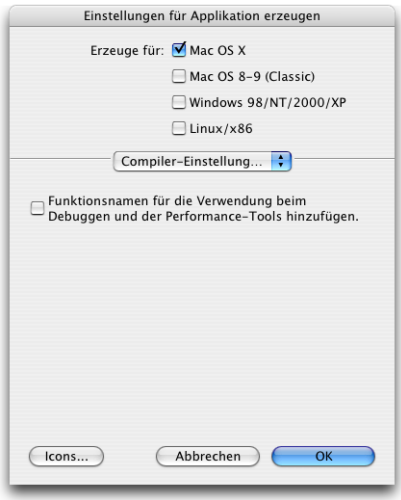
Linux-Einstellungen

Hier können Sie Ihrer Linux-Applikation einen Namen geben.



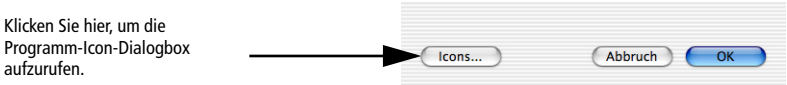
Compiler-Einstellungen

Im letzten Panel können Sie einstellen, ob die erzeugten Applikationen Funktionsnamen enthalten sollen. Diese können zu Debugging-Zwecken verwendet werden.



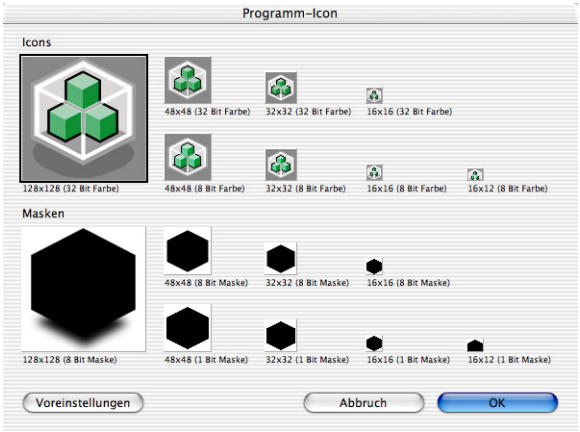
Individuelle Icons für Ihre Applikation

Über die **Compiler-Einstellungen**-Dialogbox können Sie Ihrer Applikation ein individuelles Icon zuweisen (dies funktioniert nur, wenn Ihre Anwendung einen eigenen Creator-Code besitzt). Kicken Sie dazu auf den Icons…-Knopf.



Im folgenden Dialog können Sie Icons mit verschiedenen Größen und Farbtiefen über die Zwischenablage einfügen.

Abb. 282: Die Programm-Icon-Dialog



Sie können auch Icons löschen und von einem Resource-Typ in einen anderen kopieren. Die Größenanpassung erfolgt dabei automatisch. Elemente folgender Typen können über die Zwischenablage eingefügt werden: **icns**, **icl8**, **ICN#**, **ics8**, **ics#**, **icm#**, **icm8** und **PICT**. Weisen Sie Ihrer Applikation ein individuelles Icon zu, indem Sie es in der passenden Farbtiefe und Größe in diesen Dialog einfügen. Um wieder zu den Standard-Icons zurückzukehren, klicken Sie auf **Voreinstellungen**. Nachdem Sie eine Auswahl getroffen haben, bestätigen Sie diese mit **OK**.

Damit wird das „custom icon bit" für Ihre Applikation gesetzt, so dass für diese im Finder ein 32-Bit-Farbicon angezeigt wird, auch wenn Sie keinen Creator-Code festgelegt haben. Falls Sie Ihr Programm veröffentlichen wollen, empfehlen wir Ihnen, einen Creator-Code bei Apple zu registrieren.

Hinweis: Mac OS-Versionen < 8.5 unterstützen weder 32-Bit Icons noch 128x128 Pixel- und 48x48 Pixel-Icons.

Eine alternative Möglichkeit, Ihrer Applikation individuelle Icons zuzuweisen besteht darin, die Dateiinformation des Icons (inklusive 32-Bit „icns"-Icons) in eine Resource-Datei zu packen und diese in Ihr Projekt einzufügen. Dieses Icon ersetzt dann das im Programm-Icon-Dialog festgelegte.

Eine Anwendung zum Compilieren vorbereiten

Auch wenn es sehr einfach ist, eine Stand-Alone-Applikation zu erzeugen, sollten Sie dennoch die Besonderheiten und Einschränkungen der jeweiligen Zielplattform berücksichtigen, um die Leistungsfähigkeit Ihrer Applikation zu optimieren. Der folgende Abschnitt befasst sich mit diesem Thema.

REALbasic bietet mit Remote Debugging eine komfortable Möglichkeit, Applikationen auf anderen Plattformen zu testen. Wenn Sie einen zweiten Computer besitzen, auf dem ein anderes Ziel-Betriebssystem läuft, kann der Remote-Debugger eine Testversion Ihres Programms auf diesen übertragen und dort starten.

Weitere Informationen dazu finden Sie im Abschnitt „Remote Debugging" auf Seite 364.

Für Windows compilieren

Um ein Programm für Windows zu erzeugen, müssen Sie in den Compiler-Einstellungen lediglich die Checkbox „Erzeuge für Windows…" aktivieren und dem zu erzeugenden .exe-File im Bereich „Windows-Programmeinstelllungen" einen Namen geben.

Auf dem Macintosh bekommt die Windows-Applikation ein Virtual PC™-Icon. Falls Virtual PC installiert ist, können Sie auf die erzeugte Windows-Applikation einen Doppelklick ausführen, um Virtual PC zu starten und die Applikation zu öffnen. Wenn Sie die .exe-Datei auf einen Windows-Rechner oder in ein Virtual PC-Verzeichnis kopieren, hat sie das Standard-REALbasic-Icon.

Besonderheiten bei Windows-Programmen

Für jede erzeugte Windows-Anwendung sollten Sie aus Kompatibilitätsgründen folgende Punkte prüfen:

- *Zeilenendezeichen:* Wenn Sie Texte in EditFields per Code eintragen, muss einem Carriage Return ein Line Feed folgen. Benutzen Sie dazu die EndOfLine-Funktion anstatt der Steuerzeichen selbst.
- Nicht-ASCII-Zeichen: Zeichen mit ASCII-Codes größer 127 unterscheiden sich unter Mac OS und Windows und je nach Sprache. Beispielsweise hat der Aufzählungspunkt „•“ unter Mac OS den Wert 165, unter Windows 149. Wenn Sie Text aus einer fremden Quelle lesen, können Sie bei den Methoden Read, ReadLine und ReadAll optional die Codierung angeben. REALbasic speichert die Codierung intern. Wenn Sie Text in einer bestimmten Codierung schreiben wollen, konvertieren Sie den Text vor dem Schreiben mit der ConvertEncoding-Methode.
- *MDI Interface:* Die Größe des MDI-Rahmens (des Elternfensters) ist derzeit nicht konfigurierbar. Dies sollten Sie bedenken, wenn Sie die Größe eines Fensters festlegen, das Sie öffnen möchten.
- *Windows GUI:* Halten Sie sich an die Windows-User-Interface-Richtlinien. Andernfalls sieht Ihre Applikation wie eine Portierung vom Mac auf Windows aus und nicht wie eine richtige Windows-Applikation.
- *Interface-Elemente:* Viele Windows-Interface-Elemente unterscheiden sich vom Macintosh. Ein in der Größe veränderbares Dokumentfenster hat auf dem Mac z.B. einen Vergrößerungsknopf. Unter Windows existiert dieser nicht. Wenn ein Steuerelement wegen seiner Positionierung das Vorhandensein eines Vergrößerungsknopfes voraussetzt, wird es unter Windows nicht korrekt aussehen.
- *Reihenfolge der Steuerelemente:* Die Reihenfolge der Steuerelemente ist unter Windows wichtiger, da dort auch CheckBoxes und PushButtons den Fokus bekommen können. Testen Sie daher die Reihenfolge der Steuerelemente unter Windows, bevor Sie Ihre endgültige Applikation erzeugen.
- *Multiple Document Interface:* Viele MDI-Applikationen unter Windows maximieren den MDI-Rahmen, wenn sie gestartet werden. Das MDI-Rahmenfenster ist das Elternfenster, in dem die Fenster der Applikation geöffnet werden. Wenn Sie eine Win32-Version Ihres REALbasic-Projekts compilieren und wollen, dass das MDI-Rahmenfenster beim Start maximiert wird, fügen Sie folgenden Code in den Open-Event Ihrer Application-Klasse ein:

```
Dim result as Integer
CONST SW_MAXIMIZE = 3
Declare Function ShowWindow Lib "user32" Alias "ShowWindow"
     (ByVal hwnd As Integer, ByVal nCmdShow As Integer) As Integer
Declare Function GetActiveWindow Lib "user32"
     Alias "GetActiveWindow" As Integer
result = ShowWindow(GetActiveWindow(), SW_MAXIMIZE)
```

  Hinweis: Wenn Sie keine Application-Klasse in Ihrem Projekt haben, wählen Sie den Menüpunkt **Ablage/Neue Klasse**, stellen **Application** als **Super**-Eigenschaft der neuen Klasse ein und nennen Sie **App**.
- *Mac-spezifische Steuerelemente:* Einige Steuerelemente (wie die ChasingArrows) gibt es nur auf dem Mac und haben unter Windows kein Pendant. Obwohl REALbasic eine Windows-Version dieser Steuerelemente anbietet, sollten Sie sich überlegen, ob diese in der Windows-Version Ihrer Applikation passend sind oder nicht. Die Toolbar-Steuerelemente sind nur unter Mac OS X 10.2 und höher verfügbar. Wenn Sie diese verwenden, benötigen Sie alternative Elemente für die anderen Plattformen.
- *IsCMMClicked Funktion:* Diese Funktion liefert True, wenn der Anwender die rechte Maustaste drückt. Sie funktioniert nur unter Windows und bei Steuerelementen, die einen Mousedown-Event-Handler besitzen.
- *Windows-Menüs:* Einige Standard-Menüs sind unter Windows anders benannt als unter Mac OS. Dieses Problem können Sie in REALbasic elegant mit Konstanten lösen. Mehr dazu im Abschnitt „Lokalisieren einer Applikation mit Hilfe von Konstanten“ auf Seite 208.
- *Mac-spezifische Funktionen:* Einige Fähigkeiten von REALbasic stehen nur unter Mac OS zur Verfügung, z.B. AppleEvents und AppleScript, Resource-Dateien und Shared Libraries. Mittels bedingter Compilierung können Sie Pro-

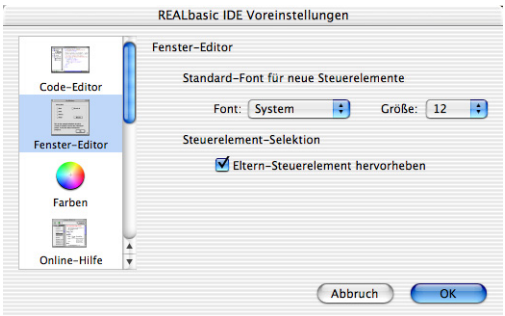
grammteile beim Erzeugen einer Windows-Anwendung ausblenden, die nur für den Mac relevant sind. Bedingte Compilierung verwendet folgendes Konstrukt:

```
#If TargetBoolean then
  //Plattformspezifischen Code hier einfügen
#Elseif TargetBoolean2 then
  //Plattformspezifischen Code für Plattform TargetBoolean2 hier einfügen
#Endif
```

TargetBoolean ist eine boole'sche Konstante oder ein boole'scher Ausdruck und ermöglicht es Ihnen, den folgenden Code nur für diesen speziellen Fall einzubinden. Folgende Konstanten stehen zur Auswahl: TargetMacOS, TargetMachO, TargetPPC, TargetCarbon, TargetLinux und TargetWin32. Mit Hilfe dieser Konstanten testen Sie, für welche Plattform die Applikation momentan erzeugt wird. Sie können auch Ausdrücke verwenden, die True oder False zurückgeben. Beispielsweise können Sie mit RBVersion feststellen, welche REALbasic-Version verwendet wird. Mehr dazu finden Sie unter „Cross-Plattform-Entwicklung" in der Sprachreferenz.

- *Mac-spezifische FolderItems:* Einige Funktionen, die Verweise auf Mac OS-spezifische Ordner zurückliefern, liefern unter Windows oder Linux Nil oder ein FolderItem zurück, das nicht dem entspricht, was Sie erwarten. Ist dies der Fall, können sie mithilfe des #if-Ausdrucks die Plattform prüfen, für die die Applikation momentan erzeugt wird.
- *Toolbox-Aufrufe:* Mit der Declare-Anweisung können Sie direkte Toolbox-Aufrufe für die Macintosh- oder die Windows-Plattform ausführen. Natürlich unterscheiden sich die Toolbox-Funktionen auf den verschiedenen Plattformen. Benutzen Sie auch hier die bedingte Compilierung, um die Declare-Anweisungen und Toolbox-Aufrufe für die verschiedenen Plattformen voneinander zu isolieren.
- *Datenbank-Datenquellen:* Unter Windows stehen derzeit nur der ODBC-Datenbanktreiber und die integrierte REAL-Datenbank zur Verfügung.
- *Schriftgrößen für Steuerelemente:* Die drei Plattformen, die von REALbasic unterstützt werden, haben ihre eigenen Vorgaben bezüglich standardmäßig vorhandener Systemzeichensätze und Schriftgrößen. Oftmals sieht ein Zeichensatz auf der einen Plattform gut aus, auf der anderen ist er jedoch zu groß oder zu klein. Aus diesem Grund besitzt die Einstellungen-Dialogbox eine Option, um die Schriftgrößen-Voreinstellung des Ziel-Betriebssytems als Standardschriftgröße für alle Steuerelemente auszuwählen, die Text verwenden. Diese Option erlaubt es Ihnen, gleichzeitig mehrere voreingestellte Schriftgrößen für Steuerelemente auszuwählen. Wählen Sie eine Schriftgröße von null, um von dieser Option Gebrauch zu machen.

Abbildung 283. Auswahl der voreingestellten Schriftgröße des jeweiligen Systems für Steuerelemente



Was Sie bei Anwendungen für Mac OS X beachten sollten

Nachdem Sie Ihre Applikation für Mac OS X erzeugt haben, sollten Sie bei allen Interface-Elementen testen, ob sie die richtige Größe haben und die Beschriftungen korrekt angezeigt werden. Außerdem muss jede Shared Library, die die Applikation verwendet, carbonisiert sein.

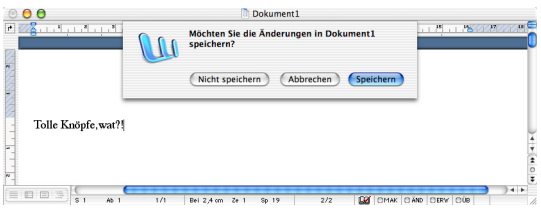
Windows-Anwender sollten sich mit den Aqua Human Interface Guidelines von Apple vertraut machen. Es handelt sich dabei um Empfehlungen von Apple zur Gestaltung von Benutzeroberflächen. Die Guidelines finden Sie unter:

`http://developer.apple.com/techpubs/macosx/Essentials/AquaHIGuidelines/`

Im folgenden werden einige grundlegende Unterschiede zwischen Windows und der Benutzeroberfläche des Macintosh behandelt. Einige Unterschiede lassen sich mit sorgfältigem Design der Benutzeroberfläche überbrücken. Andere hingegen erfordern alternative Versionen von Fenstern, Dialogen und anderen Elementen der Benutzeroberfläche. Um die verschiedenen Versionen eines Objekts zu verwalten, können Sie den *#If*-Ausdruck einsetzen. So können Sie Ihr Programm zielplattformabhängig compilieren. Information zum bedingten Compilieren finden Sie in der Sprachreferenz.

- Auf dem Macintosh gibt es nur eine Menüleiste, die sich immer am oberen Bildschirmrand befindet. Fenster haben keine eigenen Menüleisten. Auch wenn REALbasic den Einsatz von mehrerer Menüleisten für verschiedene Fenster unterstützt, steht dies nicht im Einklang mit Apples Richtlinien für die Gestaltung von Benutzeroberflächen.
- Der Windows-Menüpunk *Datei* heißt auf dem Macintosh *Ablage*. Für Probleme dieser Art bietet REALbasic eine elegante Lösung. Mehr dazu im Abschnitt „Lokalisieren einer Applikation mit Hilfe von Konstanten“ auf Seite 208.
- Unter Mac OS X befindet sich der *Einstellungen*-Menüpunkt für allgemeine Programmeinstellungen im Applikationsmenü, nicht im Datei-, Ablage-, Optionen- oder Werkzeugmenü. In REALbasic gibt es eine spezielle Klasse, die sich dieses Problems annimmt: die PrefsMenuItem-Klasse. Verwenden Sie diese Klasse an Stelle der MenuItem-Klasse für den Menüpunkt *Einstellungen*.
- Der Macintosh bietet keine Unterstützung für das Multiple Document Interface von Windows. Unter Mac OS X liegt jedes Fenster auf einer eigenen Ebene. Das bedeutet, dass mit Klick auf ein Fenster nicht gleichzeitig alle anderen Fenster der Anwendung in den Vordergrund kommen. Wenn Sie Ihre Applikation als MDI-Applikation entworfen haben, sollten Sie sie unbedingt auch unter diesem Gesichtspunkt erproben.
- Die Apple-Maus hat standardmäßig nur einen Mausknopf, wenngleich auch zahlreiche Lösungen mit mehreren Mausknöpfen von Drittanbietern auf dem Markt erhältlich sind. Das Äquivalent zum Rechtsklick unter Windows ist auf dem Macintosh der Ctrl-Klick. Stellen Sie sicher, dass Ihre Applikation keine Mausfunktionen voraussetzt, die mit der Eintastenmaus des Macs nicht möglich sind.
- Unter Mac OS X dient das Dock dazu, Applikationen und Dokumente zu starten. Über die DockItem-Eigenschaft der Application-Klasse können Sie das Programmsymbol im Dock manipulieren. Über die DockItem-Eigenschaft der Windows-Klasse können Sie das Erscheinungsbild eines Dokuments im Dock manipulieren (Wenn ein Benutzer unter Mac OS X ein Dokumentfenster minimiert, erscheint es als Icon im Dock). Die Icons sollten 128x128 Pixel groß sein.
- Die Knöpfe in Dialogboxen verwenden Verben statt „Ja“ oder „Nein“ und sind anders angeordnet. Oft ist die Reihenfolge des Abbrechen- und Bestätigen-Knopfes im Vergleich zu Windows genau vertauscht. Unten sehen Sie typische Dialoge zum Sichern von Änderungen unter Mac OS X und Windows XP. Es bestehen wesentliche Unterschiede bezüglich Beschriftung und Anordnung der Knöpfe.

Abb. 284: „Änderungen speichern“-Dialoge unter Mac OS X und Windows XP



- Apple rät von Werkzeugleisten mit vielen kleinen Icons ab und empfiehlt den Einsatz einiger weniger, qualitativ hochwertiger und größerer Icons. Die Icons sollten beschriftet sein. Solche Werkzeugleisten können mit Hilfe der ToolbarItem- und der StandardToolbarItem-Klassen erzeugt werden. Mac OS X-Applikationen verwenden außerdem Schubladenfenster, um beim Klick auf ein Icon der Werkzeugleiste weitere Auswahlmöglichkeiten anzubieten. Apple empfiehlt auch den Einsatz schwimmender Paletten als Alternative zu sehr umfangreichen Werkzeugleisten.

- Auf dem Macintosh gibt es einige standardmäßig reservierte Tastaturkürzel. Sie sollten diese nicht mit Ihrer Applikation überschreiben. Die Aqua User Interface Guidelines empfehlen auch Tastaturkürzel für häufig ausgeführte Operationen, wie z.B. für die Einträge des Datei- und Bearbeiten-Menüs.
- Während Mac OS X und Windows aus der Endung (dem Suffix) einer Datei auf deren Inhalt schließen, verwendet Mac OS 8/9 dafür so genannte Typ- und Creator-Codes. Sollen Dokumente, die mit Ihrer Applikation erzeugt wurden, unter Mac OS Classic verwendet werden, müssen Sie solche Codes erzeugen und bei Apple registrieren.
- Carbon-Applikationen beinhalten eine automatisch generierte „plst"-Resource. In dieser sind die Informationen enthalten, die in älteren Mac OS-Versionen in den „vers"-, „open"-, „FREF"-, „BNDL"- und „kind"-Resourcen zu finden waren. Wenn Mac OS X in einem Programm eine „plst"-Resource findet, verwendet es diese an Stelle der „alten" Resourcen. Sie können die automatisch erzeugte „plst" 0-Resource überschreiben, indem Sie eine eigene erzeugen, in einer Resource-Datei ablegen und diese in Ihr Projekt einfügen.

Anmerkungen zu Linux-Applikationen

Wenn Sie Applikationen für Linux erzeugen wollen, sollten Sie die folgenden Informationen beachten.

Anforderungen

Von REALbasic erzeugte Linux-Applikationen laufen nur auf x86-Computern und benötigen GTK+ 2.0 (was weitere Pakete wie GDK, Pango, Atk etc. voraussetzt). Sie können die GTK+ 2.0 Bibliotheken unter http://www.gtk.org herunterladen. Eventuell finden Sie beim Hersteller Ihrer Distribution eine vorcompilierte Version.

Grundsätzliche Informationen

Einige Steuerelemente, die unter Mac OS X und Windows funktionieren, sind unter Linux nur eingeschränkt nutzbar.

- RB3DSpace: Das RB3DSpace-Steuerelement benötigt OpenGL.
- Sounds: Benötigen libsndfile (http://www.zip.com.au/~erikd/libsndfile/). Durch „weak linking" funktioniert die Applikation aber auch, wenn diese Bibliothek nicht installiert ist. Sounds werden dann nicht abgespielt.
- Canvas: Scrollen führt dazu, dass der gesamte Canvas-Inhalt neu gezeichnet wird, falls das Canvas von einem anderen Steuerelement überlagert wird.
- Drag&Drop: Beim Draggen über Steuerelemente, die einen Mouse-Over-Effekt haben (wie PushButtons) wird die Bound-Eigenschaft nicht aktualisiert.
- EditField: SelCondense, SelExtend, SelOutline und SelShadow werden nicht unterstützt.
- FolderItem: Das Erstellungsdatum eines FolderItems kann nicht gesetzt werden.
- Menüs: Zwei Fenster können nicht auf das gleiche Menü verweisen. Wenn zwei Fenster das gleiche Menü verwenden sollen, müssen das Menü duplizieren.
- MoviePlayer und QuickTime-basierte Klassen werden derzeit nicht unterstützt.
- PushButton, CheckBox und RadioButton: Der „gedrückt"-Status dieser Steuerelemente kann nicht unterdrückt werden, wenn der MouseDown-Event True liefert.
- Scrollbars verhalten sich anders als in Mac OS X und Windows. Wenn Sie in einen der Pfeilknöpfe klicken, ohne dass LiveScroll aktiv ist, wird der ValueChanged-Event erst ausgelöst, wenn Sie den Mausknopf loslassen.
- Speak: Die Speak-Methode wird in diesem initialen Release nicht unterstützt.
- RadioButtons: In einer Gruppe von RadioButtons muss immer einer ausgewählt sein.
- Popup-Menüs: Separatoren werden nicht unterstützt.
- NotePlayer: Der NotePlayer wird in diesem initialen Release nicht unterstützt.
- Drucken: Zum Drucken wird die Bibliothek libgnomeprint 2.2 (oder höher) benötigt. Zusätzlich muss CUPS installiert sein. Es kann nur auf Postscript-Druckern gedruckt werden. Unter Linux gibt es keinen Papierformat-Dialog. Der Aufruf der PrinterSetup-Methode liefert False zurück und es wird kein Dialog angezeigt.

Vorgabesprache

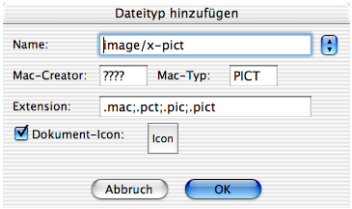
Wenn Sie Ihr Projekt mit Hilfe von Konstanten so organisiert haben, dass Sie verschiedensprachige Versionen erzeugen können, ist es möglich, die Vorgabesprache für den nächsten Compilerdurchlauf im Sprache-Popup auszuwählen. Die Default-Sprache ist die, die bei den Projekt-Einstellungen eingestellt wurde. Mehr zu diesem Thema finden Sie im Abschnitt „Lokalisieren einer Applikation mit Hilfe von Konstanten“ auf Seite 208.

Dokument-Icons

Wie Sie Ihrem Programm ein eigenes Icon geben können, wurde weiter vorn bereits erklärt.

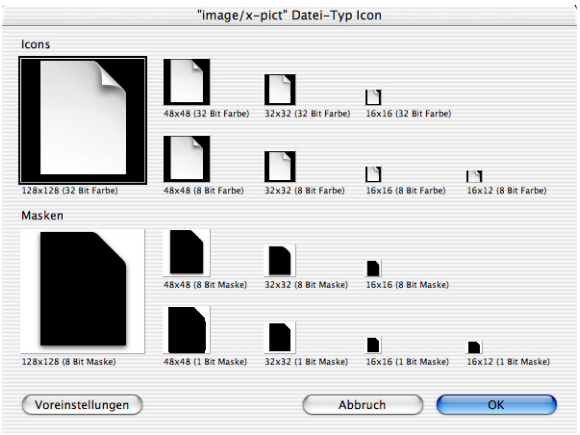
Unter **Bearbeiten/Dateitypen** können Sie auch den von Ihrem Programm erzeugten Dokumenten eigene Icons zuordnen. Öffnen Sie dazu den Dateitypen-Dialog und führen Sie einen Doppelklick auf den gewünschten Dateityp aus.

Abb. 285: Dateityp mit Dokument-Icon in der Dateitypen-Dialogbox:



Klicken Sie dann auf den **Icons**-Knopf. Damit rufen Sie die Dokument-Icons-Dialogbox auf:

Abb. 286: Die Dialogbox zum Festlegen von Dokument-Icons



Hier können Sie analog zum Vorgehen bei Programm-Icons (siehe Seite 394) aus der Zwischenablage Icons in verschiedenen Größen und Farbtiefen einfügen.

Mehrere Benutzer unter Mac OS 9

Unter Mac OS 9 können Sie über das Kontrollfeld „Mehrere Benutzer“ verschiedene Benutzer anlegen und diesen den Zugriff auf alle Applikationen oder nur auf bestimmte Programme erlauben. Wenn einem Benutzer, dem Sie das Verwenden einer mit REALbasic geschriebenen Anwendung gestattet haben, trotzdem der Zugriff verweigert wird, liegt das

daran, dass sich das Betriebssystem die Zugriffsrechte für Applikationen über deren Creator-Code merkt. Ihre Applikation benötigt also einen Creator-Code, damit sie unter Mac OS 9 korrekt mit „Mehrere Benutzer“ zusammenarbeitet.

Der Thread-Manager

Die REALbasic-Entwicklungsumgebung benötigt einen installierten Thread-Manager. Das aus Ihrem Projekt erzeugte Stand-Alone-Programm wiederum benötigt nur dann einen Thread-Manager, wenn Sie Unterklassen verwendet haben, die aus der Thread-Klasse abgeleitet wurden.

Ländercodes

Folgende Tabelle zeigt die Ländercodes, die in der „vers“-Resource einer erzeugten Applikation verwendet werden:

Abb. 287: Länder-Codes, die in der „vers”-Resource verwendet werden:

Code	Wert	Code	Wert
00	US	22	Malta
01	France	23	Cyprus
02	Britain	24	Turkey
03	Germany	25	Yugoslavia
04	Italy	33	India
05	Netherlands	34	Pakistan
06	Belgium-Lux.	36	It. Swiss
07	Sweden	40	Anc. Greek
08	Spain	41	Lithuania
09	Denmark	42	Poland
10	Portugal	43	Hungary
11	Fr. Canada	44	Estonia
12	Norway	45	Latvia
13	Israel	46	Lapland
14	Japan	47	Faeroe Isl.
15	Australia	48	Iran
16	Arabia	49	Russia
17	Finland	50	Ireland
18	Fr. Swiss	51	Korea
19	Gr. Swiss	52	China
20	Greece	53	Taiwan
21	Iceland	54	Thailand

Kapitel 15 Visual Basic-Projekte portieren

Weil es sehr viele Gemeinsamkeiten von REALbasic und Visual Basic gibt, ist es relativ einfach, aus einem Visual Basic Projekt eine Macintosh-Version unter REALbasic zu erzeugen.

Inhalt

- Importieren von Forms und Code
- Tipps, die die Konvertierung erleichtern
- Der VB Project Konverter
- Datenbank-Anbindung

Importieren von Forms und Code

REALbasic kann Forms von Visual Basic 2.0 (oder neuer) importieren und den dazugehörenden Code. REALbasic baut das Interface dazu neu zusammen und importiert den Code, der den einzelnen Steuerelementen zugeordnet ist, in die entsprechenden Event-Handler und Methoden.

Um VB-Forms zu importieren, gehen Sie folgendermaßen vor:

1. Optional für Macintosh-Anwender: Kopieren Sie die VB-Forms-Dateien (diese enden mit .frm) auf den Macintosh.
2. Öffnen Sie ein neues oder ein existierendes REALbasic-Projekt.
3. Ziehen Sie die Forms-Dateien vom Desktop in das Projektfenster, um sie zu importieren.

Weil es zwischen REALbasic und Visual Basic Unterschiede gibt, werden Fehler auftreten, die Sie zunächst korrigieren müssen. Nach dem Import der Forms-Dateien starten Sie dazu das Projekt, um dann nach und nach die aufgetretenen Probleme zu beheben.

Vereinfachen der Konvertierung

Es gibt ein paar Maßnahmen, die Sie ergreifen können, um im Vorfeld die Konvertierung von Visual Basic-Projekten nach REALbasic zu erleichtern. REALbasic wird mit einem Tool names VB Project Converter (VBPC) ausgeliefert, das die Konvertierung erleichert. Um ein Projekt zu konvertieren, ziehen Sie das VB-Projekt auf das VB Project Converter-Programm. Es werden automatisch alle Dateien, die zum Projekt gehören, eingelesen (diese müssen sich im selben Verzeichnis wie die Projektdatei selbst befinden). Formular-Ressourcen (*.frx-Dateien) müssen sich im selben Verzeichnis wie das zugehörige Formular befinden. Es ist nicht notwendig, *.frx-Ressource-Dateien manuell hinzufügen. Diese werden bei Bedarf vom VB Project Converter eingelesen.

Sie können die Dateien deselektieren, die nicht in Ihr REALbasic-Projekt übernommen werden sollen.

Bekannte Probleme

Die folgenden Probleme betreffen Unterschiede in der Architektur von VB und REALbasic.

- *Steuerelemente*: Falls ein Steuerelement in REALbasic nicht unterstützt wird, konvertiert VBPC es in ein Canvas-Steuerelement. VB-Views oder VB-Steuerelemente, die als Container für andere Steuerelemente dienen, kann VBPC nicht konvertieren, da der Parser dieser VBPC-Version ineinander verschachtelte Elemente nicht unterstützt.

Falls Sie in Ihrem VB-Projekt MonthView-Steuerelemente verwenden, werden diese durch ckMonthView-Canvas-Steuerelemente ersetzt. ckMonthView ähnelt im Wesentlichen dem Erscheinungsbild und der Funktionalität des VB-MonthView-Steuerelements.

- *Menüs*: Menüs werden nur in Formularen unterstützt. REALbasic unterstützt außer dem Standard-Tastaturkürzel nur die Wahl- und die Shifttaste. Funktionstasten werden durch die Nummer der Taste ersetzt. Die Löschen-Taste (Delete) wird durch ein „D“ ersetzt, die Einfügen-Taste (Insert) durch ein „I“, die Backspace-Taste durch ein „B“. Jede andere Spezialtaste wird entfernt.
- *MDI:* VBPC unterstützt noch keine MDI-Projekte, an der Unterstützung von MDI wird aber gearbeitet. In REALbasic wird die MDI-Unterstützung mit einer MDI-Klasse und nicht mit einem Container realisiert. In REALbasic ist es nicht möglich, Steuerelemente im MDI-Elternfenster zu verwenden.
- *Icons*: Ein Icon wird als Bild importiert und enthält das eigentliche Bild und eine Maske.
- *Konstanten*: VBPC versucht den Datentyp einer Konstanten zu erraten. Sie müssen beim Öffnen eines konvertierten Projekts in REALbasic den Datentyp jeder Konstante überprüfen.
- *Datenbank*: VBPC bietet (derzeit) nur eine eingeschränkte Datenbankunterstützung. ADAO- und DAO-Steuerelemente werden Ihrem Projekt hinzugefügt. Da VBPC Ihrem Projekt keine Datenbank zuordnen kann, ist bei Steuerelementen, die mit einer Datenbank verbunden sind, zwar die DataField-Eigenschaft korrekt gesetzt, nicht aber die DataSource-Eigenschaft. Um eine MS Access-Datenbank in eine REAL-Datenbank zu konvertieren, sollten Sie sich den Access Converter von Jose Cuevas ansehen. Datadesigners, Crystal Reports und ähnliche Produkte werden nicht unterstützt.

Datenbank-Anbindung

Visual Basic-Programme benutzen oft die Microsoft Access- (oder Jet-) Datenbank-Engine. REALbasic Standard und Professional verfügt über eine eingebaute Datenbank-Engine und eine Plugin-Schnittstelle für weitere Datenbanksysteme, die Sie verwenden können, um Datenbankapplikationen zu portieren.

Index

F

G