

Wilfried Koch

***Das C++Builder
Rezeptbuch***

Teil 1

Inhaltsverzeichnis

Vorwort.....	12
1. Arbeiten mit Formularen und einfachen Steuerelementen.....	14
1.1. Ein paar Formalien.....	14
1.1.1. Bezeichnungen der grafischen Steuerelemente.....	14
1.1.2. Die Typographie.....	15
1.2. Vor der Entwicklung des ersten Programms.....	15
1.2.1. Installation des C++Builders.....	15
1.2.2. Start des C++Builders.....	15
1.2.3. Installation des Turbo C++ Explorers.....	16
1.3. Wichtige Einstellungen.....	16
1.3.1. Editor.....	16
1.3.1.1. Tabulator und Einrückungen.....	17
1.3.1.2. Schriften und Spaltenzahl (Rechter Rand).....	17
1.3.2. Debugger.....	18
1.3.3. Verzeichniseinstellungen.....	18
1.3.3.1. Bedingungen.....	19
1.3.3.2. Include-Verzeichnis.....	20
1.3.3.3. Bibliotheks-Verzeichnis.....	20
1.3.4. Übersetzung mit oder ohne Laufzeitbibliotheken?.....	20
1.3.4.1. Auswahl der erforderliche Laufzeitbibliotheken.....	20
1.3.4.2. Probleme bei der Entwicklung auf mehreren Rechnern vermeiden.....	20
1.4. Erste Erfahrungen mit der Entwicklungsumgebung – Das „Hallo-Welt“- Programm.....	21
1.4.1. Aufgabenstellung.....	22
1.4.2. Lösung.....	22
1.4.2.1. Öffnen eines neuen VCL-C++-Projekts.....	22
1.4.2.2. Arbeiten mit dem Editor.....	23
1.4.2.3. Standardschaltflächen im Formular.....	27
1.4.2.4. Platzierung visueller Komponenten auf dem Formular - Einbau des Beschriftungsfelds (Komponente vom Typ TLabel).....	28
1.4.2.5. Namensbildung für Instanzen visueller Komponenten.....	28
1.4.2.6. Einbau der Schaltfläche (Komponente vom Typ TButton).....	28
1.4.2.7. Interaktives Erstellen von Funktionen, die auf Ereignisse antworten...29	
1.4.3. Programmcode: „Hallo-Welt“-Programm.....	31
1.4.3.1. Hauptprogramm ProHalloWelt.....	31
1.4.3.2. Hauptformular FrmMainHalloWelt.....	33

1.4.4. Unterschiede zwischen C++Builder (RAD Studio) und Turbo C++.....	35
1.4.4.1. Dateiendungen und Verzeichnisse.....	35
1.4.4.2. Programmcode: Turbo C++-Hauptprogramm ProHalloWelt.....	35
1.4.5. Erweiterung – Klicken auf das Formular.....	36
2. Grundlagen für die Gestaltung von Bedienoberflächen.....	37
2.1. Arbeiten mit zwei (oder mehr) Windows-Formularen.....	37
2.1.1. Aufgabenstellung.....	37
2.1.2. Lösung.....	37
2.1.2.1. Öffnen eines neuen VCL-C++-Projekts und Anlegen der Formulare.....	37
2.1.2.2. Einbau der Schaltfläche zum Öffnen von Form2.....	37
2.1.2.3. Verbindung zwischen den Formularen.....	37
2.1.3. Programmcode: Programm zum Arbeiten mit zwei Formularen.....	38
2.1.3.1. Hauptprogramm Pro_2Formulare.....	38
2.1.3.2. Hauptformular FrmMain_2Formulare.....	39
2.1.3.3. Zweitformular Form2_2Formulare.....	40
2.2. Anzeigen eines Begrüßungsfensters beim Programmstart.....	41
2.2.1. Lösung.....	41
2.2.1.1. Allgemeiner Teil.....	41
2.2.1.2. Fenster schließen durch Schalten.....	41
2.2.1.3. Fenster schließen nach Zeitablauf.....	42
2.2.1.4. Alternativen und Probleme.....	42
2.2.2. Programmcode: Darstellung eines Begrüßungsfensters.....	42
2.2.2.1. Programm ProBegrueessungSchalt (Fenster schließen durch Schalten).....	42
2.2.2.2. Programm ProBegrueessungZeit (Fenster schließen durch Zeitablauf).....	43
2.2.3. Dynamische Instanziierung des Begrüßungsfensters.....	44
2.3. Arrays von Steuerelementen mit einfachen Mitteln realisieren.....	44
2.3.1. Aufgabenstellung.....	45
2.3.2. Lösung.....	45
2.3.3. Programmcode: Darstellung von Arrays von Steuerelementen.....	46
2.3.3.1. Hauptfenster FrmMainIndexStat.....	46
2.4. Arrays von Steuerelementen dynamisch realisieren.....	48
2.4.1. Lösung.....	48
2.4.2. Programmcode: Dynamische Realisierung von Arrays von Steuerelementen.....	49
2.4.2.1. Hauptformular FrmMainIndexDyn.....	49
2.4.3. Übertragbarkeit auf andere Steuerelemente.....	50
2.4.3.1. Lösung.....	50
2.5. Kontextsensitive Bedienung und Information.....	51

2.5.1. Aufgabenstellung.....	51
2.5.2. Lösung.....	51
2.5.2.1. Popup-Menü.....	51
2.5.2.2. Beschriftungsfelder.....	52
2.5.2.3. Die Klasse TFont.....	53
2.5.2.4. Die Schablonenklasse Set.....	54
2.5.2.5. Ereignismethoden.....	55
2.5.3. Programmcode: Programm zur Anwendung von Pop-upmenüs.....	56
2.5.3.1. Hauptformular MainPopUp.....	56
2.6. Listen- und Kombinationsfelder.....	58
2.6.1. Eine einfache Anwendung eines Listenfeldes.....	58
2.6.2. Lösung.....	59
2.6.2.1. Laden einer Datei.....	59
2.6.2.2. Manipulation der Liste.....	59
2.6.2.3. Korrektur des Einstellbereichs des Textdrehfeldes.....	60
2.6.3. Programmcode: Anwendung von Listenfeldern.....	60
2.6.3.1. Hauptformular MainListfeld.....	61
2.6.4. Mehr Komfort durch den Einsatz eines Kombinationsfeldes.....	62
2.6.5. Lösung.....	63
2.6.5.1. Manipulation der Liste.....	63
2.6.6. Programmcode: Anwendung von Kombinationsfeldern.....	63
2.6.6.1. Hauptformular UMainKombifeld.....	63
2.7. Tabellen- (Gitter-) steuerfelder (Grids).....	65
2.7.1. Erstellung einer Rechentabelle.....	65
2.7.2. Lösung.....	66
2.7.2.1. Erstellung der Gitterüberschriften.....	66
2.7.2.2. Methoden zur Berechnung der Zeilen- und Spaltensumme.....	66
2.7.2.3. Ereignisroutine für die Änderung des Zelleninhalts.....	67
2.7.3. Programmcode: Rechentabelle für Kostenabrechnung.....	68
2.7.3.1. Hauptformular MainKalkGitter.....	68
3. Anzeigen und Bearbeiten von Textdateien.....	71
3.1. Arbeiten mit Textketten.....	71
3.1.1. Aufbau von ANSI- und Unicode-Strings.....	71
3.1.2. Wichtige Operationen für ANSI- und UnicodeStrings.....	72
3.2. Ein einfacher Texteditor.....	73
3.2.1. Aufgabenstellung.....	73
3.2.2. Lösung.....	74
3.2.2.1. Hauptfenster.....	74
3.2.2.2. Menü.....	74

3.2.2.3. Anzeigebereich für den Text.....	75
3.2.2.4. Dateiauswahlfenster.....	76
3.2.3. Weitergehende Möglichkeiten.....	77
3.2.3.1. Zeilenumbruch.....	77
3.2.3.2. Bildlauf ermöglichen.....	77
3.2.3.3. Löschen von Textabschnitten.....	77
3.2.3.4. Löschen von Zeilen.....	77
3.2.3.5. Einfügen von Zeilen.....	77
3.2.3.6. Einstellen des Zeichensatzes.....	78
3.2.4. Drucken.....	78
3.2.4.1. Druckereinrichtung.....	78
3.2.4.2. Drucken.....	78
3.2.5. Programmcode: Einfacher Texteditor.....	79
3.2.5.1. Hauptformular FormMainEdit.....	79
4. Modularisierung von Bedienoberflächen.....	83
4.1. Frames als Mittel der Programmgliederung.....	83
4.2. Innerhalb ein und desselben Formulars mehrere ähnliche Baugruppen mit Frames realisieren.....	85
4.2.1. Aufgabenstellung.....	85
4.2.2. Lösung.....	85
4.2.2.1. Erstellen eines Frames.....	86
4.2.2.2. (Wieder-) Verwendung eines bereits existierenden Frames.....	86
4.2.3. Programmcode: Realisierung ähnlicher Baugruppen in einem Formular mittels Frames.....	86
4.2.3.1. Frame FrameAnzeige.....	86
4.2.3.2. Hauptformular MainFormAnzeigen2Mal.....	87
4.2.4. Erweiterung der Aufgabenstellung.....	89
4.2.5. Lösung.....	89
4.2.6. Programmcode: Erweiterte Aufgabenstellung für die Anwendung von Frames.....	90
4.2.6.1. Frame FrameAnzeigen.....	90
4.2.6.2. Hauptformular MainFormFrameZaehler.....	90
4.2.7. Einsatz des Debuggers.....	91
4.2.7.1. Verfolgung des Programmablaufs.....	91
4.2.7.2. Untersuchung und Manipulation von Daten.....	92
4.2.7.3. Bedingte Haltepunkte	95
4.2.7.4. Debuggergerechtes Codieren.....	97
4.3. Frame als an mehreren Orten eines Programms genutzte Zentralinstanz.....	97
4.3.1. Aufgabenstellung.....	97

4.3.2. Lösung.....	98
4.3.3. Programmcode: ProFrame1Instanz.....	98
4.3.3.1. Formular FormMain.....	98
4.3.3.2. Formular FormPanel.....	99
4.4. Frames als wiederverwendbare Programmbausteine	100
4.5. Gereiterte Steuerelemente (TPageControl, TTabControl).....	101
4.5.1. Aufgabenstellung.....	102
4.5.2. Lösung.....	102
4.5.2.1. Lösung mit TPageControl und TTabSheet.....	102
4.5.2.2. Lösung mit TTabControl.....	103
4.5.2.3. Resultatsausgabe.....	104
4.5.3. Programmcode: Beispiel für die Anwendung gereiteter Steuerelemente.....	104
4.5.3.1. Hauptformular FrmMainTabPage.....	104
4.5.3.2. Frame FraAusgabe.....	108
4.6. MDI-Oberfläche.....	109
4.6.1. Aufgabenstellung.....	109
4.6.2. Lösung.....	110
4.6.2.1. MDI-Oberfläche und Menü.....	111
4.6.2.2. Aktionslisten.....	111
4.6.2.3. Schaltflächenleiste.....	113
4.6.2.4. Bilderliste.....	113
4.6.2.5. Das MDI-Kind.....	114
4.6.2.6. Speichern.....	115
4.6.2.7. Bearbeiten.....	116
4.6.2.8. Fensteranordnung.....	116
4.6.2.9. Schließen.....	117
4.6.2.10. Beenden.....	117
4.6.3. Programmcode: Programm mit MDI-Oberfläche.....	117
4.6.3.1. Hauptformular FrmMDIMain.....	117
4.6.3.2. MDI-Kind FrmMDIChild.....	121
5. Variable Bedienoberflächen.....	123
5.1. Bildschirm mit Panels in mehrere größenverstellbare Bereiche einteilen.....	123
5.1.1. Entwicklung mit dem Formular-Editor.....	123
5.1.1.1. Teilerleisten.....	123
5.1.1.2. Steuerelemente auf den Panels.....	124
5.1.2. Programmcode: Programm mit variabel eingeteilter Bedienoberfläche.....	124
5.1.3. Dynamisches Anlegen der Komponenten.....	125
5.2. Verschieben von visuellen Elementen mit der Ziehen-und-Ablegen- (Drag-And-Drop-) Technologie.....	126

5.2.1. Aufgabenstellung.....	127
5.2.2. Lösung.....	127
5.2.3. Programmcode: Drag-and-Drop-Beispiel.....	128
5.2.3.1. Hauptformular FrmMainDragAndDrop.....	128
6. Baumdarstellungen.....	131
6.1. Klassen für die Baumdarstellung.....	131
6.2. Aufgabenstellung.....	131
6.3. Lösung.....	131
6.3.1. Bearbeiten des Baums.....	131
6.3.2. Speichern und Laden von Bäumen.....	132
6.4. Programmcode: Programm für die Baumdarstellung.....	133
6.4.1. Hauptformular FrmMainBaum.....	133
6.4.1.1. Schnittstellendatei UFrMMainBaum.h.....	133
6.4.1.2. Implementationsdatei UFrMMainBaum.cpp.....	134
7. Grafische Darstellungen.....	136
7.1. Malen und Zeichnen.....	137
7.2. Bitmaps bearbeiten leicht gemacht.....	137
7.2.1. TImage die Schlüsselkomponente zur Bilddarstellung.....	138
7.2.1.1. TImage und Canvas.....	138
7.2.2. Arbeiten mit Bitmaps.....	139
7.2.2.1. Laden und Speichern von Bitmaps.....	139
7.2.2.2. Drucken von Bitmaps.....	140
7.2.2.3. Zugriff auf einzelne Pixel von Bitmaps	140
7.2.3. Aufgabenstellung.....	142
7.2.4. Lösung.....	142
7.2.4.1. Das Hauptformular FrmMainBitmapMan.....	142
7.2.4.2. Das Ergebnisformular für die Filterung Frm2BitmapMan.....	143
7.2.5. Programmcode: Manipulation von Bitmap-Daten.....	145
7.2.5.1. Das Hauptformular FrmMainBitmapMan.....	145
7.2.5.2. Das Hilfsformular Frm2BitmapMan.....	148
7.3. Vektorgrafik – Ein einfaches Zeichenprogramm.....	149
7.3.1. Aufgabenstellung.....	149
7.3.2. Lösung.....	150
7.3.2.1. Fachklasse Zeichnung.....	150
7.3.2.2. Benutzerschnittstellen-Klasse FrmMainZeichnung.....	152
7.3.3. Programmcode: Erstellung von Zeichnungen (Vektorgrafiken).....	153
7.3.3.1. Hauptformular FrmMainZeichnung.....	153

7.3.3.2. Zeichnungsmodul Zeichnung.....	156
Literaturverzeichnis.....	160
Stichwortverzeichnis.....	160

```

    Application->ShowException(&exception);
}
catch (...)
{
    try
    {
        throw Exception("");
    }
    catch (Exception &exception)
    {
        Application->ShowException(&exception);
    }
}
return 0;
//-----

```

2.2.3. Dynamische Instanziierung des Begrüßungsfensters

Eine Variationsmöglichkeit besteht in der dynamischen Instanziierung des Begrüßungsfensters. In seinem Konstruktor wird das Hauptfenster zunächst auf unsichtbar geschaltet. Danach wird das Begrüßungsfenster instanziiert und modal angezeigt. Nach dem Schließen des Begrüßungsfensters wird das Hauptfenster angezeigt und das Begrüßungsfenster wird mit der Methode *Release* frei gegeben. *Release* gibt das Formular erst dann frei, wenn dessen eigene Ereignisbehandlungsrouninen sowie die seiner untergeordneten Komponenten beendet sind. Die Steuerung wird jedoch sofort an die aufrufende Routine zurückgegeben. Verwenden Sie im Zusammenhang mit Formularen immer *Release* statt *Free* oder *Delete*. *Release* funktioniert auch bei Anwendung auf nicht instanziierte Objekte.

Formulare
immer mit
Release frei-
geben!

Die Schnittstellendatei des Begrüßungsfensters (**UFrmBegrueung.h**) muss in diesem Fall in die Implementationsdatei des Hauptfensters (**UFrmMain.h**) eingebunden werden.

```

fastcall TFrmMain::TFrmMain(TComponent* Owner)
: TForm(Owner)
{
    Visible = false;
    TFrmBegrueung * FrmBegrueung = new TFrmBegrueung(FrmMain);
    FrmBegrueung->ShowModal();
    FrmBegrueung->Release();
    Visible = true;
}

```

2.3. Arrays von Steuerelementen mit einfachen Mitteln realisieren

Häufig verwendet man in Anwendungen Steuerelemente gleichen Typs, die in Gruppen angeordnet sind. Legt man diese Steuerelemente mit Bordmitteln des C++Builders oder von Turbo C++ an, so erhalten sie die Bezeichner *Control_n* bis *Control_{n+j}*

Wertanzeige mittels der Menüfunktion Auswerten / Ändern

Lesefehler

Abbildung 4.4: Wertanzeige- und -änderung per Menüfunktion

Mittels der Menüfunktion **Start|Auswerten/Ändern** oder  -  kann eine ähnliche Funktion aufgerufen werden. Dabei wird die Variable ausgewertet, auf der der Cursor steht. Sind mehrere Zeichen angewählt (was meist keinen Sinn macht!), dann wird genau die angewählte Zeichenfolge als Bezeichner ausgewertet. Abbildung 4.4 zeigt das dieser Menüfunktion zugeordnete Formular. Das erste Textfeld (*Ausdruck*) gibt den auszuwertenden Ausdruck wieder. Das Feld *Ergebnis* zeigt dessen aktuellen Wert und *Neuer Wert* im dritten Feld zeigt den Wert an, auf den der Ausdruck – falls es sich dabei um eine Variable handelt – geändert werden soll. Die Übernahme der Änderung erfolgt mit der Schaltfläche **Ändern**. Betätigung der Schaltfläche **Untersuchen** erfolgt zum Übergang auf den nachfolgend beschriebenen Menüpunkt.

Wertanzeige mittels der Menüfunktion Untersuchen

Ähnliche Wertedarstellungen wie im Debugger (Abbildung 4.3) erhält man mit der Menüfunktion **Untersuchen**. Diese können Sie direkt (**Start|Untersuchen**) oder wie in Abbildung 4.4 gezeigt über eine Schaltfläche aus der Funktion **Auswerten/Ändern** erreichen. Es öffnet sich dann ein Überwachungsfenster, wie es in den Abbildungen 4.6 und 4.7 dargestellt ist. Dieses eröffnet mittels dreier Karteikarten den Zugang zu den Attributen (Daten), Methoden und Eigenschaften des überwachten Objekts.

Vielfach stellt eine Zeile in diesen Karteikarten ein komplexes Objekt dar (wie zum Beispiel *VertScrollBar* eine vertikale Scrollbar. Wenn Sie auf diese Zeile klicken öffnet sich ein weiteres Fenster (4.7) in dem Details der vertikale Scrollbar betrachtet werden können.

2.7. Tabellen- (Gitter-) steuerfelder (Grids)

Tabellensteuerfelder dienen der Darstellung von Informationen in Tabellen- oder Matrixstruktur. C++Builder liefert hierfür eine Abstrakte Klasse `TCustomGrid` und zwei konkrete Klassen `TStringGrid` und `TDrawGrid`.

Lesefehler

Abbildung 2.6: Mit einem Tabellenelement vom Typ `TStringGrid` implementierte Rechentabelle

2.7.1. Erstellung einer Rechentabelle

Es soll eine Rechentabelle erstellt werden, in der Aufwendungen im Zusammenhang mit Reisen verwaltet und aufaddiert werden können. Die Tabelle besitzt 14 Zeilen (Zeile 0 bis 13) und 10 Spalten (Spalte 0 bis 9). Die Zeilen stehen für die Spaltenüberschriften (Zeile 0), die Monatsumsätze (Zeilen 1 bis 12) und die Jahressummen (Zeile 13). Die Spalten stehen für die Zeilenüberschriften (Spalte 0), die einzelnen Sachkonten (Spalte 1 bis 8) und die Monatssummen (Spalte 9).

Während des Betriebes sollen die Zeile 0 und die Spalte 0 nicht geändert werden können. Ebenso sollen die Zeile 13 und die Spalte 9 nur als Ergebnis einer Berechnung nicht aber manuell geändert werden können.

Der Bereich von Spalte 1, Zeile 1 (Zelle [1][1]) bis Zelle [8][12] (Arbeitsbereich) wird zunächst mit Nullen vorbesetzt. Davon leiten sich die Monatssummen in den Zeilen 1 bis 12 der Spalte 9 und die Jahressummen in den Spalten 1 bis 8 der Zeile 13 ab. Weiter natürlich die Gesamtsumme in Zelle [9][13]. Alle diese Summen besitzen zunächst ebenfalls den Wert 0.

In den Arbeitsbereich können beliebige Zahlenwerte eingetragen werden. Unmittelbar nach dem Eintrag eines gültigen Zahlenwerts werden die Monats und Jahressummen aktualisiert, wobei nur die erforderlichen Berechnungen durchzuführen sind. Ungültige Werte werden kommentarlos abgelehnt. Alle Zahlenwerte werden als Festkomma-

zahlen mit 2 Nachkommastellen dargestellt. Der Dezimalseparator soll sich dabei nach dem jeweils gültigen Gebietsschema richten.

2.7.2. Lösung

Öffnen Sie eine VCL-Formularanwendung, bezeichnen Sie das Hauptformular mit *MainKalkGitter* und ziehen Sie ein Gitterelement vom Typ *TStringGrid*, das Sie *StGKalkGitter* nennen, auf das Hauptformular.

2.7.2.1. Erstellung der Gitterüberschriften

Damit die Zeile 0 und die Spalte 0 während des Betriebes sollen nicht geändert werden können, setzen Sie mittels des Objektinspektors die Werte der Eigenschaften *FixedRows* und *FixedCols* auf 1. Außer wie in unserem Beispiel zur Entwurfszeit kann diese Festlegung auch zur Laufzeit erfolgen. Dass die Zeile 13 und die Spalte 9 nur als Ergebnis einer Berechnung nicht aber manuell geändert werden können, muss dann per Programm sicher gestellt werden.

Die Festlegung der Gitterüberschriften erfolgt im Konstruktor des Hauptformulars mittels folgender Anweisungen:

```
StGKalkGitter->Cells[1][0] = "Porto";
.....
StGKalkGitter->Cells[9][0] = "Monatssumme";
StGKalkGitter->Cells[0][0] = "Monat\\Konto";
StGKalkGitter->Cells[0][1] = "Januar";
.....
StGKalkGitter->Cells[0][12] = "Dezember";
StGKalkGitter->Cells[0][13] = "Jahressumme";
```

Erster Index:
Spalte,
zweiter Index:
Zeile

Bitte beachten Sie die Indizierung der Zellen: Der erste Index für die Zellen bezeichnet die Spalte, der zweite die Zeile.

2.7.2.2. Methoden zur Berechnung der Zeilen- und Spaltensumme

Für die Berechnung der Zeilen- bzw. Spaltennummern werden die Methoden *float Spaltensumme(int iSpalte)* und *float Zeilensumme(int iZeile)* vorgesehen.

Spalten- bzw. Zeilennummer werden als Parameter übergeben. Die Berechnung erfolgt in einer Schleife über alle relevanten Zellen einer Spalte (Zeilen 1 bis 12) oder einer Zeile (Spalten 1 bis 8).

Die Resultate (Summen) werden als Funktionswert übergeben.

6. Baumdarstellungen

In der betrieblichen Praxis sind viele Informationen baumförmig strukturiert: Als typisches Beispiel hierfür sind jedem Programmierer z. B. die Verzeichnisbäume der Dateiverzeichnisse bekannt. Derartige Daten sollten daher auch in Programmen baumförmig implementiert und visualisiert werden.

6.1. Klassen für die Baumdarstellung

Für die Implementation von Bäumen gibt es im C++Builder die Klasse `TTreeNodees` (Plural!!). Der einzelne Knoten in Bäumen dieser Klasse wird mittels der Klasse `TTreeNode` implementiert. Die Klasse `TTreeView` dient der Visualisierung dieser Bäume. Bäume können zur Entwurfszeit oder programmgesteuert erstellt werden.

6.2. Aufgabenstellung

Zu erstellen ist ein Programm mittels dessen Bäume interaktiv erstellt werden können. Die Bäume sollen außerdem auf Datenträger gespeichert und vom Datenträger wieder hergestellt werden können. Vereinfachend seien die Knoten der Bäume ausschließlich Texte. Auf die Realisierung komplexerer Knoten wird im Beispiel verzichtet.

6.3. Lösung

Zur Darstellung des Baumes wird eine Komponente des Typ `TTreeView` auf das Hauptformular gezogen. Die Bearbeitung des Baumes wird über vier Schaltflächen und ein Textfeld ermöglicht. Weiterhin wird für die statische Beschriftung (Erläuterung) ein Beschriftungsfeld hinzugefügt

Außerdem werden je zwei Schaltflächenpaare zum Speichern und Laden des Baums auf dem Formular untergebracht. Über das eine Schaltflächenpaar wird das Speichern und Laden als Datei, über das andere das Speichern und Laden als Stream gesteuert.

6.3.1. Bearbeiten des Baums

Die vier oben erwähnten Schaltflächen dienen

- dem Anhängen eines Knotens am Ende der jeweiligen Anwahlebene,
- dem Einfügen eines Knotens vor dem angewählten Knoten,
- dem Anlegen eines Kindknotens und
- dem Löschen eines Knotens.

Bezugselement ist dabei immer der momentan angewählte Knoten. Das Textfeld dient zur Eingabe des Namens für den neuen Knoten. Weitere Daten werden den Knoten nicht zugeordnet.

- **Anhängen** fügt auf der gleichen Ebene wie der angewählte Knoten unter demselben Vaterknoten ein Element hinzu.
- **Einfügen Vor** fügt ein Element direkt vor dem angewählten Knoten ein.
- **Kind hinzufügen** fügt ein Element direkt unter dem angewählten Knoten ein.

Die Wirkung dieser drei Funktionen ist in den Abbildungen 6.1 bis 6.4 dargestellt.

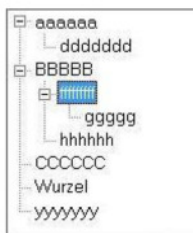


Abbildung 6.1:
Ausgangszu-
stand

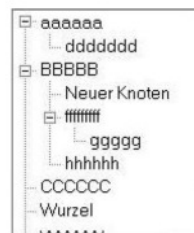


Abbildung 6.2:
Funktion
Einfügen vor

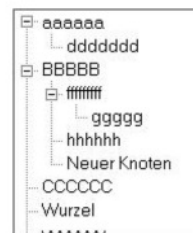


Abbildung 6.3:
Funktion
Anhängen

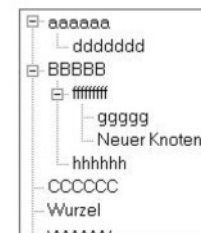


Abbildung 6.4:
Funktion
Kindknoten
einfügen

6.3.2. Speichern und Laden von Bäumen

Bäume können mit einfachen Mitteln, d. h. mit Methoden von `TTreeView` entweder als Dateien oder als Streams gespeichert werden. Die Grundkonstruktion für das Speichern als Datei ist folgendermaßen aufgebaut:

```
if (SaveDialog1->Execute())
{
    TreeView1->SaveToFile (SaveDialog1->FileName)
}
```

Speichert man als Stream, dann muss man die Grundkonstruktion noch um die Verbindung mit der physikalischen Datei ergänzen.

Eigentlich sollte jetzt alles funktionieren, Tests mit dem C++Builder 2009 ergeben jedoch, dass der geladene Baum nicht mit dem unmittelbar zuvor gespeicherten identisch ist¹⁴. Die Ursache hierfür liegt in einem Fehler, der wohl im Zuge der Umstellung auf UnicodeStrings in das C++Builder-IDE geraten ist. In [EMDIS1] wird eine Lösung zur Umgehung des Fehlers angegeben die im Codebeispiel in 6.4.1.2. eingebaut wurde.

Fehler in
C++Builder
2009 beach-
ten!!