

Wilfried Koch

***Professionelles Programmieren
von Anfang an
mit Free Pascal
und der freien Entwicklungsumgebung
Lazarus***

Teil 2

Inhaltsverzeichnis (Übersicht)

Inhaltsverzeichnis.....	6
Vorwort.....	18
19. Gestaltung fortgeschrittener Bedienungsoberflächen zum Zweiten.....	20
20. Baumdarstellungen.....	65
21. Modularisierung von Bedienoberflächen.....	78
22. Eingriffe in den Programmablauf.....	157
23. Anspruchsvolle Geschäftsgrafiken problemlos erstellen (Einsatz des TACart- Rahmenwerks).....	181
24. Nutzen Sie die Möglichkeiten Dynamischer Link-Bibliotheken (DLLs).....	235
25. Indy öffnet die Tür zum Internet.....	269
26. Am Ende ist nicht alles vorbei (Teil 2) - Fortgeschrittene Persistenzlösungen / Arbeiten mit Datenbanken.....	334
Stichwortverzeichnis.....	434
Quellenverzeichnis.....	444

Inhaltsverzeichnis

Inhaltsverzeichnis (Übersicht).....	5
Vorwort.....	18
19. Gestaltung fortgeschrittener Bedienungsoberflächen zum Zweiten.....	20
19.1. Arbeiten mit zwei (oder auch mehr) Windows-Formularen.....	20
19.1.1. Aufgabenstellung.....	20
19.1.2. Der Beitrag von Lazarus.....	20
19.1.3. Lösung.....	20
19.1.3.1. Öffnen eines neuen Lazarus-Projekts und Anlegen der Formulare.....	20
19.1.3.2. Einbau der Schaltflächen zum Öffnen von Form2_2Formulare.....	20
19.1.3.3. Verbindung zwischen den Formularen.....	21
19.1.4. Programmcode: Programm zum Arbeiten mit zwei Formularen.....	21
19.1.4.1. Hauptprogramm Pro_2Formulare.....	21
19.1.4.2. Hauptformular FrmMain_2Formulare.....	22
19.1.4.3. Zweitformular Frm2_2Formulare.....	24
19.2. Anzeigen eines Begrüßungsfensters beim Programmstart.....	25
19.2.1. Aufgabenstellung.....	25
19.2.2. Der Beitrag von Lazarus.....	25
19.2.3. Lösung.....	25
19.2.3.1. Allgemeiner Teil.....	25
19.2.3.2. Begrüßungsfenster durch Schalten schließen.....	26
19.2.3.3. Begrüßungsfenster nach Zeitablauf schließen.....	26
19.2.3.4. Alternativen und Probleme.....	26
19.2.4. Programmcode: Darstellung eines Begrüßungsfensters.....	27
19.2.4.1. Programm ProBegrueassungSchalt (Fenster schließen durch Schalten).....	27
19.2.4.2. Programm ProBegrueassungZeit (Fenster schließen durch Zeitablauf).....	27
19.2.4.3. Formular FrmBegrueassung.....	28
19.2.5. Dynamische Instanziierung des Begrüßungsfensters.....	29
19.3. Arrays von Steuerelementen mit einfachen Mitteln realisieren.....	29
19.3.1. Aufgabenstellung.....	30
19.3.2. Der Beitrag von Lazarus.....	30
19.3.3. Lösung.....	30
19.3.4. Programmcode: Darstellung von Arrays von Steuerelementen.....	31
19.3.4.1. Hauptfenster FrmMainIndexStat.....	31
19.4. Arrays von Steuerelementen dynamisch realisieren.....	33
19.4.1. Lösung.....	33

19.4.2. Programmcode: Dynamische Realisierung von Arrays von Steuerelementen	34
19.4.2.1. Hauptformular FrmMainIndexDyn.....	34
19.4.3. Übertragbarkeit auf andere Steuerelemente / Dynamische Realisierung von Steuerelementen in mehreren Hierarchiestufen.....	35
19.4.3.1. Aufgabenstellung.....	35
19.4.3.2. Der Beitrag von FreePascal und Lazarus.....	35
19.4.3.3. Lösung.....	37
19.4.3.4. Quellcode des Programms zur dynamischen Erzeugung hierarchisch aufgebauter Steuerelemente.....	40
19.5. Kontextsensitive Bedienung und Information.....	44
19.5.1. Aufgabenstellung.....	44
19.5.2. Der Beitrag von Lazarus.....	44
19.5.3. Lösung.....	45
19.5.3.1. Menüeditor.....	45
19.5.3.2. Entwicklung des Popup-Menüs.....	47
19.5.3.3. Beschriftungsfelder.....	49
19.5.3.4. Die Klasse TFont.....	50
19.5.3.5. Ereignismethoden.....	51
19.5.4. Programmcode: Programm zur Anwendung von Popupmenüs.....	52
19.5.4.1. Hauptformular FrmMainPopUp.....	52
19.6. Listen- und Kombinationsfelder.....	54
19.6.1. Eine einfache Anwendung eines Listenfeldes.....	54
19.6.2. Der Beitrag von Lazarus.....	55
19.6.3. Lösung.....	56
19.6.3.1. Laden einer Datei.....	56
19.6.3.2. Manipulation der Liste.....	58
19.6.3.3. Korrektur des Einstellbereichs des Textdrehfeldes.....	59
19.6.4. Programmcode: Anwendung von Listenfeldern.....	60
19.6.4.1. Hauptformular FrmMainListFeld.....	60
19.6.5. Mehr Komfort durch den Einsatz eines Kombinationsfeldes.....	62
19.6.6. Lösung.....	63
19.6.6.1. Manipulation der Liste.....	63
19.6.7. Programmcode: Anwendung von Kombinationsfeldern.....	63
19.6.7.1. Hauptformular FrmMainKombifeld.....	63
20. Baumdarstellungen.....	65
20.1. Aufgabenstellung.....	65
20.2. Beitrag von Lazarus.....	65
20.2.1. Klassen für die Baumdarstellung.....	65

20.2.1.1. Streams – Kurzeinführung.....	66
20.2.1.2. Speichern und Laden von Bäumen.....	68
20.3. Lösung.....	73
20.3.1. Bearbeiten des Baums.....	73
20.4. Programmcode: Programm für die Baumdarstellung.....	74
20.4.1. Hauptprogramm ProBaum.....	74
20.4.2. Hauptformular FrmMainBaum.....	75
21. Modularisierung von Bedienoberflächen.....	78
21.1. Das Angebot von Lazarus: Frames als Mittel der Programmgliederung.....	78
21.2. Innerhalb ein und desselben Formulars mehrere ähnliche Baugruppen mit Frames realisieren.....	78
21.2.1. Aufgabenstellung.....	78
21.2.2. Lösung.....	78
21.2.2.1. Erstellen eines Frames.....	79
21.2.2.2. Ereignisbehandlungsmethoden für Frames und Subkomponenten von Frames.....	82
21.2.3. Programmcode: Realisierung ähnlicher Baugruppen in einem Formular mittels Frames.....	84
21.2.3.1. Frame FraAnzeigen.....	84
21.2.3.2. Hauptformular FrmMainAnzeigen2Mal.....	85
21.2.4. Erweiterung der Aufgabenstellung.....	86
21.2.5. Lösung.....	86
21.2.6. Programmcode: Erweiterte Aufgabenstellung für die Anwendung von Frames.....	87
21.2.6.1. Frame FraAnzeigenZaehl.....	87
21.2.6.2. Hauptformular FrmMainFrameZaehler.....	88
21.3. Frame als an mehreren Orten eines Programms genutzte Zentralinstanz.....	89
21.3.1. Aufgabenstellung.....	90
21.3.2. Lösung.....	90
21.3.3. Programmcode: Frame als Zentralinstanz.....	90
21.3.3.1. Hauptprogramm ProFraMehr.....	90
21.3.3.2. Hauptformular FrmMainMehr.....	91
21.3.3.3. Formular FrmEinfach.....	92
21.3.3.4. Formular FrmPanel.....	93
21.3.3.5. Frame TFraMehr.....	93
21.4. Frames als wiederverwendbare Programmbausteine.....	94
21.5. Reite(r)n, nicht nur zu Pferde - Gereiterte bzw. mehrseitige Steuerelemente (TPageControl, TTabControl, TNotebook und TExtendedNotebook).....	95

21.5.1. Aufgabenstellung.....	95
21.5.2. Der Beitrag von Lazarus.....	97
21.5.2.1. Lösungsbasis TPageControl.....	97
21.5.2.2. Lösungsbasis TTabControl.....	97
21.5.2.3. Lösungsbasis TNotebook.....	97
21.5.2.4. Lösungsbasis TExtendedNoteBook.....	98
21.5.3. Lösung.....	98
21.5.3.1. Lösung mit TPageControl und TTabSheet.....	98
21.5.3.2. Lösung mit TTabControl.....	100
21.5.3.3. Lösung mit TNotebook und TPage.....	101
21.5.3.4. Lösung mit TExtendedNoteBook und TTabSheet.....	101
21.5.3.5. Resultatsausgabe.....	101
21.5.4. Programmcode: Beispiel für die Anwendung gereiteter bzw. mehrseitiger Steuerelemente.....	102
21.5.4.1. Hauptprogramm ProMehrseitig.....	102
21.5.4.2. Hauptformular FrmMainMehrseitig.....	102
21.5.4.3. Frame TFraAus.....	107
21.6. LMDI-Anwendungen.....	109
21.6.1. Aufgabenstellung und Beitrag von Lazarus.....	109
21.6.2. Bausteine für eine Lösung.....	110
21.6.2.1. Vorarbeiten.....	110
21.6.2.2. Start der Programmentwicklung.....	112
21.6.2.3. Menüs.....	113
21.6.2.4. Menü und (L)MDI-Oberfläche.....	114
21.6.2.5. Werkzeugleiste (TToolBar).....	115
21.6.2.6. Bilderliste.....	118
21.6.2.7. Aktionslisten und Aktionen.....	120
21.6.2.8. TMultiDoc.....	124
21.6.2.9. Ein bisschen mehr Objektorientierte Programmierung.....	124
21.6.2.10. Das MDI-Kind.....	126
21.6.2.11. Schaltflächenleiste (TButtonsBar).....	128
21.6.3. Realisierung der Programmfunktionen.....	128
21.6.3.1. Anlegen eines neuen Kindformulars.....	128
21.6.3.2. Öffnen und Anzeigen einer bestehenden Datei.....	129
21.6.3.3. Öffnen einer Datei mit einer Standardanwendung.....	130
21.6.3.4. Speichern einer Datei unter dem aktuell bestehenden Namen.....	131
21.6.3.5. Speichern einer Datei unter einem neuen Namen.....	131
21.6.3.6. Speichermethoden im Hauptformular.....	132
21.6.3.7. Zusätzliche Methoden und Eigenschaften.....	133
21.6.3.8. Bearbeiten.....	134
21.6.3.9. Suchen / Suchen und ersetzen.....	136

21.6.3.10. Kindformulare schließen.....	137
21.6.3.11. Anwendung schließen.....	138
21.6.3.12. Anordnung der Fenster.....	138
21.6.4. Programmcode: Programm mit LMDI-Oberfläche.....	140
21.6.4.1. Hauptprogramm ProLMDIBeispiel.....	140
21.6.4.2. Hauptformular FrmLMDIBeispiel.....	141
21.6.4.3. Kindformular mit Textinhalt FrmLMDIChildText.....	154
21.6.4.4. Kindformular mit Grafikinhalt FrmLMDIChildGraph.....	155
22. Eingriffe in den Programmablauf.....	157
22.1. Abbruch einer laufenden Berechnung zu einem beliebigen Zeitpunkt.....	157
22.1.1. Aufgabenstellung.....	157
22.1.2. Erster Lösungsversuch.....	157
22.1.2.1. Bedienoberfläche.....	157
22.1.2.2. Berechnungsschleife.....	157
22.1.2.3. Abbruchmechanismus.....	158
22.1.2.4. Testergebnis.....	159
22.1.3. Lösung.....	159
22.1.3.1. Schritthaltende Resultatsanzeige.....	159
22.1.3.2. Zeitnaher Berechnungsabbruch durch Schaltflächenbetätigung.....	160
22.1.3.3. Berechnungsabbruch mittels Tastatureingabe.....	160
22.1.3.4. Zusammenfassung.....	161
22.1.4. Programmcode: Abbruch von Berechnungen (ProAbbruch).....	162
22.1.4.1. Hauptformular FormMainAbbruch.....	162
22.1.5. Reduzierung der Prozessorbelastung.....	164
22.2. Abbruch einer laufenden Berechnung nach Ablauf einer vorgegebenen Zeit (Realisierung eines Timeouts).....	165
22.2.1. Aufgabenstellung.....	165
22.2.2. Lösung.....	165
22.2.2.1. Bedienoberfläche.....	165
22.2.2.2. Berechnungsschleife.....	166
22.2.2.3. Zeitlicher Abbruchmechanismus.....	166
22.2.2.4. Reduzierung der Prozessorbelastung.....	166
22.2.3. Programmcode: Programmabbruch durch Zeitablauf (ProTimeOut).....	166
22.2.3.1. Hauptformular FrmMainTimeOut.....	166
22.3. Laufende Anzeige von Datenänderungen.....	168
22.3.1. Aufgabenstellung.....	168
22.3.2. Elementare Lösung.....	168
22.3.2.1. Anwendung und Bedienoberfläche in der selben Unit.....	168
22.3.2.2. Verteilung von Anwendung und Bedienoberfläche auf zwei verschiedene Units.....	169

22.3.3. Systematische Lösung.....	171
22.3.3.1. Der Lösungsbeitrag von FreePascal/Lazarus.....	171
22.3.3.2. Grundlagen der Lösung.....	171
22.3.3.3. Bedienoberfläche.....	172
22.3.3.4. Anwendungsmodul.....	172
22.3.3.5. Ereignisbehandlung im Hauptformular.....	174
22.3.4. Programmcode: Ereignisgesteuerte Anzeige veränderlicher Daten.....	175
22.3.4.1. Hauptformular FrmMainEreignis.....	175
22.3.4.2. Unit UAnwendung.....	176
22.4. Gleiche Fachaufgabe – geändertes Gesicht.....	178
22.4.1. Lösung zur variierten Aufgabenstellung.....	178
22.4.2. Programmcode zur variierten Aufgabenstellung.....	179
22.4.2.1. Hauptformular FrmMainEreignisVar.....	179
23. Anspruchsvolle Geschäftsgrafiken problemlos erstellen (Einsatz des TACHart- Rahmenwerks).....	181
23.1. Aufgabenstellung.....	183
23.1.1. Fachaufgabe.....	183
23.1.2. Aufbau und Vorgehen.....	183
23.2. Lösung mit dem TACHart-Rahmenwerk.....	183
23.2.1. Das Diagramm und seine Bausteine.....	185
23.2.1.1. Wichtige Eigenschaften und Methoden des Diagramms (TChart).....	185
23.2.2. Achsen setzen den Maßstab.....	185
23.2.2.1. Achsenliste.....	185
23.2.2.2. Achsen.....	187
23.2.3. Datenreihen.....	191
23.2.3.1. Wichtige Eigenschaften und Methoden von Datenreihen (TChartSeries)	191
23.2.3.2. Wichtige Eigenschaften und Methoden von Liniendiagrammen (TLineSeries).....	191
23.2.3.3. Wichtige Eigenschaften und Methoden von Scatterplots (TManhattanSeries).....	191
23.2.3.4. Wichtige Eigenschaften und Methoden von Kreisdiagrammen (TPieSeries).....	192
23.2.3.5. Wichtige Eigenschaften und Methoden von Blasendiagrammen (TBubbleSeries).....	192
23.2.4. Datenquellen.....	193
23.2.4.1. Listenquelle (TListChartSource).....	193
23.3. Programm zur Darstellung der Wahlergebnisse.....	194
23.3.1. Programmaufbau.....	194

23.3.2. Formulare.....	195
23.3.2.1. Hauptformular FrmChartMain.....	195
23.3.2.2. Diagrammformulare FrmDiaStat (Bundestagswahl) und FrmDiaDyn (Landtagswahl Bayern).....	195
23.3.3. Allgemeine Elemente der Diagrammgestaltung.....	196
23.3.4. Darstellung der Wahlergebnisse im Dialogverfahren.....	198
23.3.5. Darstellung der Wahlergebnisse durch Codierung.....	204
23.3.5.1. Diagrammaufbau.....	204
23.3.5.2. Weitere Diagrammarten.....	205
23.3.5.3. Diagrammexport in Dateien.....	209
23.3.5.4. Speichern in der Zwischenablage.....	210
23.3.5.5. Diagrammdruck.....	210
23.3.6. Hilfsroutinen für die Darstellung der Wahlergebnisse.....	213
23.3.7. Programmcode: Erstellung von Geschäftsgrafiken.....	217
23.3.7.1. Hauptformular FrmMainWahlen.....	217
23.3.7.2. Formular FrmDiaStat.....	218
23.3.7.3. Formular FrmDiaDyn.....	222
23.3.7.4. Hilfsroutine für die Differenzbildung.....	232
24. Nutzen Sie die Möglichkeiten Dynamischer Link-Bibliotheken (DLLs).....	235
24.1. Ein bisschen Theorie.....	235
24.1.1. Möglichkeiten der DLL.....	235
24.1.1.1. Nutzung einer DLL durch mehrere Programme.....	236
24.1.1.2. Realisierung hybrider Programmkonzepte.....	236
24.1.1.3. Skalierung von Programmen.....	236
24.1.1.4. Verbesserung der Wartbarkeit.....	236
24.2. So erstellen Sie eine DLL.....	237
24.2.1. Aufgabenstellung.....	237
24.2.2. Die Lösung.....	237
24.2.2.1. Die DLL – ein Lazarus-Projekt.....	237
24.2.2.2. Schnittstellen der DLL.....	239
24.2.3. Monolithische DLL.....	240
24.2.3.1. Programmcode für die monolithische DLL.....	240
24.2.4. Modulare DLL.....	241
24.2.4.1. Programmcode für die modulare DLL.....	242
24.2.5. Implizite Bindung der DLL.....	243
24.2.5.1. Projekt ProMainImpl_D.....	243
24.2.5.2. Hauptprogramm ProDLLMainStatTest_D.....	243
24.2.5.3. Probleme beim Verbinden von EXE-Datei und DLL.....	243
24.2.5.4. Hauptformular UFrMain_D - Direkteinbau der Importschnittstelle.....	244

24.2.5.5. Hauptformular UFrmMain_Z - Zentrale Anordnung der Importschnittstelle in einer Schnittstellendatei.....	245
24.2.6. Testen von DLL-Funktionen.....	246
24.2.7. Trennung der Datenbereiche.....	246
24.2.8. Explizite Bindung.....	247
24.2.8.1. Aufbau des Hauptprogramms (DLL-benutzendes Programm).....	247
24.2.8.2. Schnittstellenmodul UDLLImp.....	247
24.2.8.3. ProgrammCode für die Dynamische Einbindung von DLLs.....	249
24.3. Auch das geht: DLLs und LCL.....	251
24.3.1. Aufgabenstellung.....	252
24.3.1.1. Realisierung eines Formulars mittels einer DLL.....	252
24.3.1.2. Hauptprogramm ProDLLmitLCL.....	252
24.3.2. Lösungsweg.....	253
24.3.2.1. Die DLL DLLmitLCL zur Realisierung des Formulars.....	253
24.3.2.2. Hauptprogramm.....	254
24.3.3. Programmcode: Verwendung von LCL-Komponenten in DLLs.....	254
24.3.3.1. DLL DLLMitLCL.....	254
24.3.3.2. Hauptprogramm (Nutzer der DLL DLLmitLCL).....	258
24.4. Noch ein Beispiel: DLL mit Objektschnittstelle.....	259
24.4.1. Aufgabenstellung.....	260
24.4.2. Lösung.....	260
24.4.2.1. Basisklasse TMWSt0.....	260
24.4.2.2. DLL DLLmitObj.....	261
24.4.2.3. Hauptprogramm ProMitObjDLL.....	261
24.4.3. Programmcode: DLL zur Kapselung von Klassen.....	263
24.4.3.1. DLL DLLMitObj.....	263
24.4.3.2. Basisklasse TMWSt0.....	264
24.4.3.3. Hauptprogramm ProDLLMitObj.....	264
24.4.3.4. Hauptformular FrmMainDLLMitObj.....	264
24.5. Zugang zu fremden Sprachwelten: Nutzung von DLLs, die in C++ verfasst wurden.....	266
24.5.1. Die modifizierte C++-DLL.....	266
24.5.2. Das Hauptprogramm.....	267
24.5.2.1. Formular FrmMainDLLMonoTest.....	267
25. Indy öffnet die Tür zum Internet.....	269
25.1. Übersicht über die Aufgabenstellungen.....	269
25.2. Was bietet mir Lazarus - Die Indy-Bibliothek.....	269
25.2.1. Geschichte und Inhalt.....	269
25.2.2. Laden und Installieren von Paketen mit dem Paket-Manager.....	271

25.3. Spezielle Indy-Datentypen.....	274
25.4. Ohne SSL geht (fast) nichts mehr – Sichere Datenübertragung mit SSL.....	277
25.5. E-Mail aus einer Anwendung senden (Entwicklung eines SMTP-Clients).....	277
25.5.1. Aufgabenstellung.....	277
25.5.2. Lösungsweg.....	280
25.5.2.1. Sendemethode BtnSMTPSendenClick.....	282
25.5.3. Programmcode: Senden von E-Mail-Nachrichten (SMTP-Client).....	285
25.5.3.1. Hauptprogramm ProSMTPClientSSL.....	285
25.5.3.2. Formularunit UFormMainSMTPClientSSL.....	285
25.6. E-Mail in einer Anwendung empfangen (Entwicklung eines POP3-Clients)....	289
25.6.1. Aufgabenstellung.....	289
25.6.2. Lösungsweg.....	291
25.6.2.1. Auflistung der eingegangenen Nachrichten.....	293
25.6.2.2. Anzeige der Textnachrichten und Auflistung der Anhänge.....	298
25.6.2.3. Aktivitätsanzeige.....	301
25.6.2.4. Abspeichern der Nachrichten und Anhänge.....	301
25.6.2.5. Löschen von Nachrichten.....	303
25.6.3. Programmcode: Empfangen von E-Mail-Nachrichten (POP3-Client).....	303
25.6.3.1. Hauptprogramm ProPOP3Client.....	303
25.6.3.2. Formularunit UFormMainPOP3ClientSSL.....	304
25.7. Dateiverkehr mit dem Internet (Erstellung eines FTP-Clients).....	309
25.7.1. Was ist FTP?.....	309
25.7.2. Aufgabenstellung.....	310
25.7.3. Installation eines Testservers.....	311
25.7.3.1. Installation von Filezilla.....	311
25.7.3.2. Installation von FreeFTPd.....	313
25.7.3.3. Installation von The Personal FTP Server 8.4f.....	316
25.7.4. Vorgehen bei der Lösung.....	316
25.7.5. Entwicklungsstufe 1.....	318
25.7.5.1. Bedienoberfläche.....	318
25.7.5.2. Verbinden.....	318
25.7.5.3. Anmelden (LogIn).....	319
25.7.6. Entwicklungsstufe 2.....	322
25.7.6.1. Bedienoberfläche.....	322
25.7.6.2. Darstellung der Verzeichnisliste.....	322
25.7.7. Entwicklungsstufe 3.....	323
25.7.8. Entwicklungsstufe 4.....	325
25.7.8.1. Bedienoberfläche.....	325
25.7.8.2. Download.....	327
25.7.8.3. Upload.....	327

25.7.8.4. Ereignisbehandlung.....	328
25.7.9. Programmcode: Realisierung eines FTP-Clients.....	328
26. Am Ende ist nicht alles vorbei (Teil 2) - Fortgeschrittene Persistenzlösungen /	
Arbeiten mit Datenbanken.....	334
26.1. Die Architektur von Datenbankanwendungen.....	335
26.2. Der Beitrag von FreePascal und Lazarus.....	336
26.2.1. Benutzerschnittstelle.....	336
26.2.2. Datenquelle.....	337
26.2.3. Datenmenge.....	337
26.2.4. Datenbankverbindung.....	337
26.2.5. Datenbanktabelle.....	337
26.2.6. Datensatz.....	337
26.3. Aufgabenstellung - Erstellung eines Programms zur Rechnungserstellung.....	338
26.3.1.1. Speicherung der Stammdaten.....	338
26.3.1.2. Speicherung der Warendaten.....	338
26.3.1.3. Speicherung der Kundendaten.....	339
26.3.2. Datentechnische Erstellung der Rechnung.....	339
26.3.3. Druckfähige Aufbereitung der Rechnungen.....	339
26.4. Grundsätzliches zu Datenbanken.....	340
26.4.1. Zugang zur Datenbank.....	340
26.4.2. Tabellen und deren Spalten.....	340
26.4.3. Schlüssel.....	341
26.5. Schaffung der Rahmenbedingungen für die Programmerstellung.....	342
26.5.1. Technologieauswahl.....	342
26.5.2. Installation des Datenbankverwaltungssystems.....	342
26.5.3. Inbetriebnahme der Datenbankverwaltungssoftware.....	343
26.6. Erstellung und Bearbeitung von Datenbanken.....	345
26.6.1. Die Datenbankabfragesprache SQL (Structured Query Language).....	346
26.6.2. Interaktive Erstellung und Bearbeitung von Datenbanken mit phpMyAdmin	
.....	348
26.6.2.1. Interaktive Erstellung der Datenbank fakturierung.....	348
26.6.2.2. Interaktive Erstellung von Datenbanktabellen am Beispiel der	
Datenbanktabelle kunden.....	349
26.6.2.3. Interaktive Bearbeitung einer Datenbank.....	350
26.6.2.4. Export und Import von Datenbanken und Datenbanktabellen.....	351
26.6.3. Erstellung von Datenbanken mittels individueller Programmierung.....	352
26.6.3.1. Erstellung der Datenbank fakturierung und der Datenbanktabelle waren	
zur Laufzeit mittels eines individuell erstellten Programms.....	352

26.6.3.2. Programmgesteuerte Erstellung der Datenbank fakturierung.....	353
26.6.3.3. Programmgesteuerte Bearbeitung einer Datenbank.....	364
26.6.4. Programm zur interaktiven Erfassung der Warenstammdaten.....	365
26.6.4.1. Hauptprogramm ProWarenBearb_SQL.....	365
26.6.4.2. Hauptformular FrmMainWarenBearb_SQL.....	365
26.7. Alle Funktionen der Rechnungserstellung unter einem Dach.....	369
26.7.1. Das Steuerelement TDBNavigator.....	371
26.7.2. Das Steuerelement TDBGrid.....	373
26.7.3. Verknüpfung von Datenbanktabellen.....	374
26.7.4. Sicherung der referenziellen Integrität.....	376
26.7.4.1. Zum Begriff der referenziellen Integrität.....	376
26.7.4.2. Ein einfacher technischer Lösungsansatz.....	377
26.7.4.3. Beispielimplementation.....	378
26.7.5. Wichtiges zu den fünf Units.....	379
26.7.5.1. UFrmMainFakturierungGesamt.....	379
26.7.5.2. UFraAllg.....	380
26.7.5.3. UFraKunden und UFraWaren.....	381
26.7.5.4. UFraRechnungen.....	381
26.7.6. Programmcode für das Programm zur Rechnungserstellung	
ProFakturierungGesamt.....	382
26.7.6.1. Programmcode des Hauptprogramms.....	382
26.7.6.2. Programmcode des Hauptformulars FrmMainFakturierungGesamt....	383
26.7.6.3. Programmcode des Frames FraAllg für die Datenbankverbindung.....	385
26.7.6.4. Programmcode des Frames UFraKunden für die Verwaltung der	
Kundendaten.....	388
26.7.6.5. Programmcode des Frames UFraWaren für die Verwaltung der	
Warendaten.....	391
26.7.6.6. Programmcode des Frames UFraRechnungen für die Verwaltung der	
Rechnungsdaten.....	394
26.8. Nicht-SQL-Datenbanken.....	402
26.8.1. Aufgabenstellung.....	402
26.8.2. Beiträge von FreePascal und Lazarus.....	402
26.8.3. Dateiformate.....	403
26.8.3.1. Mem-Format.....	404
26.8.3.2. Bin-Format.....	404
26.8.3.3. CSV-Format.....	404
26.8.3.4. SDF-Format.....	404
26.8.3.5. Fixed Format.....	406
26.8.3.6. DBF-Format.....	406
26.8.3.7. Unterschiede zu SQL.....	406
26.8.4. Entwicklung der Lösung.....	406

26.8.5. Programmcode.....	407
26.8.5.1. Hauptprogramm ProWarenDBSdf.....	407
26.8.5.2. Hauptformular FrmMainWarenDBSdf.....	407
26.8.5.3. Hauptprogramm ProWarenDBFix.....	409
26.8.5.4. Hauptformular FrmMainWarenDBFix.....	410
26.9. Datenbankberichte.....	413
26.9.1. Installation von LazReport.....	413
26.9.2. Programm zur Erstellung eines Datenbankberichts.....	415
26.9.3. Vorbereitung der Daten.....	415
26.9.4. Der Berichtseditor.....	415
26.9.5. Aufbau des Datenbankberichts / Bänder.....	416
26.9.6. Verbindung von Datenbank und Bericht.....	418
26.9.6.1. Syntax der Einträge im Rechteckobjekt.....	419
26.9.6.2. Vorschau des Berichts.....	420
26.9.6.3. Durchführung von Berechnungen im Bericht.....	420
26.9.6.4. Formatierung von Berichtsdaten.....	424
26.9.7. Aufbau der Rechnung auf der Basis von LazReport.....	424
26.9.7.1. Angaben zum Verkäufer.....	425
26.9.7.2. Angaben zum Kunden.....	425
26.9.7.3. Angaben zur Leistung und zu den Kosten.....	425
26.9.8. Verbindung von Programm, Datenbank und Bericht.....	425
26.9.8.1. Die Klasse TfrReport.....	425
26.9.8.2. Zugriff auf die Datenbank.....	425
26.9.8.3. Gestaltung des Berichts im Rahmen einer selbst erstellten Anwendung.	426
26.9.8.4. Anzeigen des Berichts.....	426
26.9.8.5. Drucken des Berichts.....	427
26.9.8.6. Speichern und erneutes Laden des Berichts.....	428
26.9.8.7. Export des Berichts in menschlesbaren Formaten.....	428
26.9.9. Programmcode für die Ausgabe des Berichts.....	429
26.9.9.1. Hauptformular FrmMainReportRechnung.....	429
26.9.9.2. Programmcode des Hauptprogramms ProReportRechnung.....	433
Stichwortverzeichnis.....	434
Quellenverzeichnis.....	444

Vorwort

Jetzt halten Sie den zweiten Teil von „Professionelles Programmieren von Anfang an“ in Ihren Händen

Wie schon der erste Teil, wendet sich auch dieses Buch sowohl an Ein- als an Umsteiger. Dabei folgt es demselben klaren didaktischen Konzept wie der erste Teil. Anhand typischer Beispielaufgaben werden Schritt für Schritt Lernziele festgelegt, um das Know-How weiter zu entwickeln. Innerhalb einer jeden Aufgabe wird dargestellt, welche Eigenschaften von Pascal und Lazarus für die Bearbeitung der Aufgabe von Bedeutung sind und wie man sie einsetzen sollte. Schließlich wird die komplette Lösung der jeweiligen Aufgabe vorgestellt.

Dabei werden u. a. folgende Themen behandelt:

- Fortgeschrittene Gestaltung von Bedienoberflächen.
- Baumdarstellungen
- Erstellung von Grafiken
- Dynamische Link-Bibliotheken (DLLs)
- Elementare Internetanwendungen
- Datenbankprogrammierung

Nach dem Studium dieses Buches sind Sie in der Lage ein erweitertes Spektrum von Aufgaben zu lösen, die durch die Sprachdefinition von (Free-) Pascal abgedeckt sind. Anders als wenn Sie sich ausschließlich auf Free-Pascal stützen würden, werden Ihre Programme von Anfang an mit einer professionellen grafischen Bedienoberfläche versehen sein, die sich auf die Bibliothek LCL (Lazarus Component Library) stützt. LCL ist auf Sprachebene (Pascal) weitgehend schnittstellenkompatibel mit der von Delphi verwendeten VCL (Visual Component Library) schnittstellenkompatibel, sodass Delphi und Lazarus-Know-How in vielen Fällen austauschbar wird.

Neu in Zusammenhang mit diesem Buch ist das Angebot der Links auf den Service-Seiten

Aktuelle Informationen zu diesem Buch, wie Korrekturen, Antworten auf Leserfragen oder auch Bestellmöglichkeiten für CDs mit dem Programmcode zu meinen Büchern finden Sie im Internet unter

www.informatik-ganz-einfach.de bzw. www.okomedien.de

Während der Arbeit an einem Buch wie diesem, müssen viele andere Dinge zurückstehen. Aus diesem Grund danke ich meiner Frau Ruth für ihre Geduld.

Für Hinweise und Anregungen zu meinen Büchern bin ich immer dankbar. Ich denke auch schon daran, einen dritten Band herauszugeben. Ein paar Ideen dazu habe ich be-

NEU:

Link-Liste auf
den Service-
Seiten zum
Buch

Was möchten
Sie in einem
eventuellen
Folgeband fin-
den?

reits. Gern erfahre ich auch von Ihnen, welche Themen Sie in einem eventuellen weiteren Band finden möchten.

Übrigens: Freie Software darf im Rahmen der Lizenzen zwar kostenlos eingesetzt werden. Ihre Entwicklung ist jedoch nicht ohne finanziellen Aufwand möglich. So ist es für ihre Nutzer nur ein Gebot der Fairness, sich an ihrer Entwicklung ggf. auch finanziell zu beteiligen. Und sei es nur mit einem kleinen Betrag. Das gilt insbesondere dann, wenn Sie die Software in kommerziellen Projekten einsetzen. Deutlich billiger als die Lizenz eines vergleichbaren proprietären Produkts ist das allemal. Förderung von Lazarus und FreePascal ist über die „FreePascal and Lazarus Foundation“ möglich. Unter <https://foundation.freepascal.org/> finden Sie hierzu mehr.

Auch freie
Software kostet!

Sie brennen sicher schon darauf, Ihre Arbeit mit Lazarus fortzusetzen. Laden Sie – falls noch nicht geschehen - die neueste Version der Integrierten Entwicklungsumgebung Lazarus aus dem Internet herunter. Installieren und starten Sie Lazarus und arbeiten Sie sich dann Schritt für Schritt anhand des Lehrtexts und der Beispiele vorwärts.

Ich wünsche Ihnen eine erfolgreiche Einarbeitung in die weitergehenden Möglichkeiten von Lazarus (und FreePascal) und anschließend eine erfolgreiche Projektarbeit!

Oberkochen im Mail 2020

Wilfried Koch

19. Gestaltung fortgeschrittener Bedienungsoberflächen zum Zweiten

19.1. Arbeiten mit zwei (oder auch mehr) Windows-Formularen

19.1.1. Aufgabenstellung

Ein Programm soll nach dem Start ein Hauptformular *FrmMain_2Formulare* öffnen. Dieses Hauptformular soll zwei Schaltflächen enthalten, mittels derer ein weiteres Formular *Frm2_2Formulare* geöffnet wird.

Das Öffnen soll so erfolgen, dass danach entweder beide Formulare aktiv sind (nicht modales Anzeigen) oder so, dass nur das zuletzt geöffnete Formular bearbeitet werden kann (modales Anzeigen). Eine Nachweismöglichkeit hierfür soll vorgesehen werden.

19.1.2. Der Beitrag von Lazarus

Lazarus ermöglicht die Verwendung beliebig vieler Formulare innerhalb einer Anwendung. Damit deren Zusammenspiel funktioniert, müssen einige einfache Regeln beachtet werden, die im folgenden Abschnitt beschrieben werden.

19.1.3. Lösung

19.1.3.1. Öffnen eines neuen Lazarus-Projekts und Anlegen der Formulare

Das Öffnen des Projekts und das Anlegen des ersten Formulars *FrmMain_2Formulare* erfolgt in bekannter Weise (siehe Teil 1, Abschnitt 9.4.1., Seite 225). Mit der Anwahl Datei|Neues Formular wird dann das zweite Formular *Frm2_2Formulare* angelegt.

19.1.3.2. Einbau der Schaltflächen zum Öffnen von Form2_2Formulare

Ziehen Sie aus der Werkzeugleiste zwei Schaltflächen (Komponenten vom Typ `TButton`) auf das Formular *FrmMain_2Formulare*. Sie erhalten die Namen `BtnModal` und `BtnNichtModal`. Beschriften Sie die Schaltflächen mittels des Objektinspektors mit `Modal öffnen` und `Nichtmodal öffnen`. Mittels der Klick-Routinen dieser Schaltflächen öffnen Sie *Form2_2Formulare*. Detailliert betrachtet erfolgt das Öffnen von *Frm2_2Formulare* mittels der Methoden `Show` oder `ShowModal` von *Frm2_2Formulare*.

Soll *FrmMain_2Formulare* auch dann ansprechbar sein, wenn *Frm2_2Formulare* geöffnet ist, so muss es mit der Methode *Show* nicht modal geöffnet werden. Möchten Sie *FrmMain_2Formulare* sperren, während *Frm2_2Formulare* aktiv ist, dann müssen Sie die Methode *ShowModal* zum modalen Öffnen verwenden.

Ob die beabsichtigte Funktion erfüllt ist, kann z. B. durch Eingabe von Zeichen in ein Textfeld und sofortige Ausgabe in ein Beschriftungsfeld nachgewiesen werden. Zu diesem Zweck wird auf jedem der beiden Formulare ein Textfeld (TEdit) und ein Beschriftungsfeld (TLabel) untergebracht.

Funktionsnachweis durch Kombination aus Beschriftungsfeld und Textfeld.

Im Falle der modalen Anzeige des zweiten Formulars sind Eingaben nur in *Frm2_2Formulare* möglich. Bei nicht modaler Anzeige sind Eingaben auf beiden Formularen möglich.

Die Sofortige Übernahme des aktuellen Textes aus dem Textfeld ins Beschriftungsfeld erfolgt durch Nutzung des Change-Ereignisses. Sobald sich der Text im Textfeld ändert wird der aktuelle Text neu ins Beschriftungsfeld übernommen.

19.1.3.3. Verbindung zwischen den Formularen

Ruft man, wie es laut der Aufgabenstellung vorgesehen ist, innerhalb der Klasse *TFrmMain_2Formulare* die Methode *Show* (oder *ShowModal*) des Objekts *Form2_2Formulare* auf (Anweisung *Form2_2Formulare.Show* bzw. *Frm2_2Formulare.ShowModal*), so muss das Objekt *Form2_2Formulare* in der Klasse *TFormMain_2Formulare* der Unit *UFrmMain_2Formulare* bekannt sein. Dieses bekannt Machen erfolgt dadurch, dass man in der Unitdatei *UFrmMain_2Formulare.pas* des Hauptfensters in einer uses-Klausel die Unit *UForm_2Formulare* aufführt. Da die Unit nur importiert wird, ist eine Nennung im Implementationsteil ausreichend. Vergessen Sie dies, dann hat das einen Linker-Fehler zur Folge.

Das zweite Formular muss dem ersten bekannt gemacht werden.

Im Gegensatz dazu braucht die Unit *UFrmMain_2Formulare* in der Unit *UForm2_2Formulare* nicht zu bekannt zu sein.

19.1.4. Programmcode: Programm zum Arbeiten mit zwei Formularen

19.1.4.1. Hauptprogramm *Pro_2Formulare*

Projektdatei *Pro_2Formulare.lpi*

Die Endung *.lpi* steht für Lazarus Project Information. Diese Datei enthält u.a. Informationen über die Bausteine des Projekts und für die Einstellung und den augenblicklichen Zustand der Integrierten Entwicklungsumgebung wie z. B. Einstellungen von Compiler und Linker. *Pro_2Formulare.lpi* wird in unserem Falle vollständig durch die

Integrierte Entwicklungsumgebung (IDE) erstellt. Die Informationen in lpi-Dateien werden im XML-Format gespeichert.

Implementationsdatei Pro_2Formulare.lpr

Die Endung **lpr** steht für Lazarus Program File. Diese Datei enthält das Hauptprogramm als Pascal-Code. Sie wird in unserem Falle durch die Integrierte Entwicklungsumgebung erstellt und weist jetzt zwei Konstruktoraufrufe (einen pro Formular) auf.

Per Konvention ist das zuerst mit *CreateForm* erstellte Formular das Hauptformular. Es wird bei Ausführung der Methode *Application.Run* ggf. als erstes angezeigt.

Das zuerst mit *CreateForm* erstellte Formular wird Hauptformular.

```
program Pro_2Formulare;

{$mode objfpc}{$H+}

uses
  {$IFDEF UNIX}{$IFDEF UseCThreads}
  cthreads,
  {$ENDIF}{$ENDIF}
  Interfaces, // this includes the LCL widgetset
  Forms, UFrMMain_2Formulare, UFrM2_2Formulare
  { you can add units after this };

{$R *.res}

begin
  RequireDerivedFormResource:=True; //Jedes Formular muss resource haben.
  Application.Initialize;
  Application.CreateForm(TFrMMain_2Formulare, FrmMain_2Formulare);
  Application.CreateForm(TFrM2_2Formulare, Frm2_2Formulare);
  Application.Run;
end.
```

19.1.4.2. Hauptformular FrmMain_2Formulare

Implementationsdatei UFrMMain_2Formulare.pas

```
unit UFrMMain_2Formulare;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils, FileUtil, Forms, Controls, Graphics, Dialogs, StdCtrls
  ;

type

  { TFrMMain_2Formulare }
```

20. Baumdarstellungen

In datentechnischen Aufgabenstellungen sind viele Informationen baumförmig strukturiert: Als typisches Beispiel hierfür sind jedem Programmierer die Verzeichnisbäume der Dateiverzeichnisse bekannt. Derartige Daten sollten daher auch in Programmen als Baumstrukturen implementiert und visualisiert werden.

20.1. Aufgabenstellung

Erstellen Sie ein Programm, mittels dessen Bäume interaktiv erstellt werden können. Die Bäume sollen außerdem auf Datenträger gespeichert und vom Datenträger wieder hergestellt werden können. Vereinfachend bestehe die Knoteninformation der Bäume ausschließlich aus Texten. Auf die Realisierung komplexerer Knoten wird im Beispiel verzichtet.

20.2. Beitrag von Lazarus

20.2.1. Klassen für die Baumdarstellung

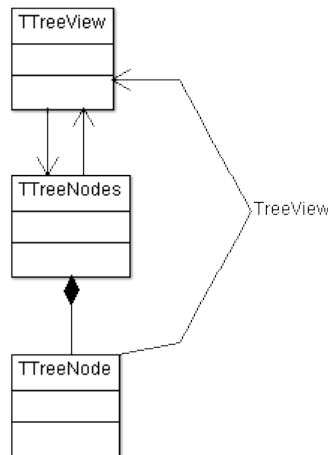


Abbildung 20.1: Zusammenhang zwischen den Klassen *TTreeView*, *TTreeNodes* und *TTreeNode*. Zur Symbolik s.a. Abbildung 19.1.

Für die Implementation von Bäumen gibt es in Lazarus die Klasse `TTreeNodes` (Plural!!). Der einzelne Knoten in Bäumen dieser Klasse wird mittels der Klasse

Unterschied
`TTreeNodes`
vs.
`TTreeNode`
beachten!!!

`TTreeNode` (Singular!!) implementiert. Die Klasse `TTreeView` [LAZTREEV] dient der Visualisierung dieser Bäume. Bäume können zur Entwurfszeit oder programmgesteuert (z. B. unter der Nutzung von Streams) erstellt werden. Die Tabellen 20.4 bis 20.7 zeigen die Möglichkeiten dieser Komponenten.

20.2.1.1. Streams – Kurzeinführung

Streams sind eine angemessene Möglichkeit zur Speicherung von Bäumen. Sie werden hier nur ansatzweise behandelt. Eine Vertiefung des Themas ist für einen Folgeband vorgesehen.

Mit Streams (Datenströmen) kann zunächst einmal alles realisiert werden, was mit Files (Dateien) möglich ist. Hinzu kommt, dass mit Streams auch komplexe Datenmengen – z. B. komplette Objekte – mit einfachen Mitteln übertragen werden können.

In vielen Klassen, so z. B. auch für die Klasse `TTreeView` stellt Lazarus Methoden zum Streamen zur Verfügung. Wenn Sie Streaming-Prozeduren selbst implementieren möchten, können Sie auf die Klasse `TStream` und deren Nachfolger zurückgreifen.

Lazarus unterscheidet das Streamen in:

- Arbeitsspeicherbereiche,
- Dateien und
- Textketten.

Für diese Anwendungsfälle gibt es die von `TStream` (teilweise indirekt) abgeleiteten Klassen `TBytesStream`, `TMemoryStream`, `TFileStream` und `TStringStream`.

Von diesen vier Streamtypen speichert lediglich `TFileStream` die Informationen persistent ab. Aus diesem Grund kommt nur dieser Streamtyp für eine dauerhafte, programmunabhängige Datenspeicherung in Betracht.

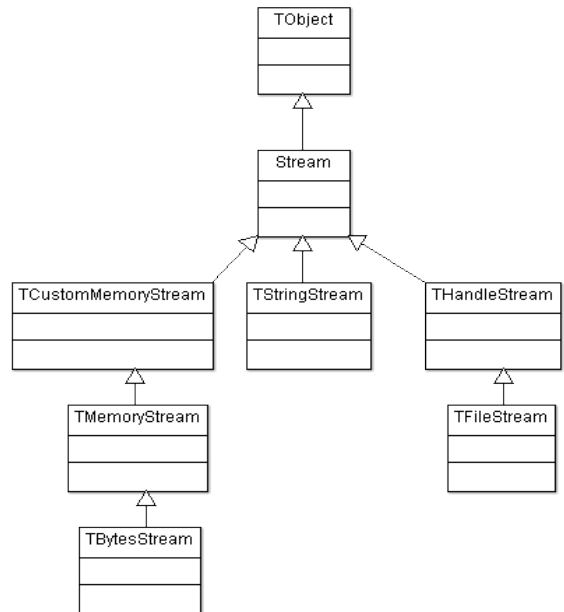


Abbildung 20.2: Klassendiagramm der Stream-Klassen von Lazarus. Zur Symbolik s. a. Abbildung 19.1

TFileStream (unit Classes)	
Eigenschaft	Bedeutung
FileName : string;	(Disk-) Datei in der der Stream gespeichert wird.
Position : Int64;	Gegenwärtige Position im Stream
Size: Int64;	Gegenwärtige Größe des Streams
Methode	Bedeutung
function CopyFrom (Source: TStream; Count Int64) : Int64;	Kopiert Count Bytes aus Source in den aktuellen Stream; Funktionswert: tatsächlich übertragene Bytezahl; Count = 0 bedeutet, dass die ganze Datei übertragen wurde. Die Byte- anzahl erhalten Sie dann in der Eigenschaft Size (s. o.).
Constructor Create (const AFileName: String, Mode: Word);	Erstellt eine neue Instanz von TFileStream. Was genau ge- schieht, wird mittels des Parameters Mode festgelegt. S. a. fmOpenxxx in Tabelle 20.3.
Destructor Destroy;	Zerstört den Stream.
procedure Seek (Offset: LongInt; Origin: Word);	Offset: Versatz der Startposition in Bytes Origin: Bezugspunkt für die Suche. Siehe soFromXXX in Ta- belle 20.3.

Tabelle 20.2: Wichtige Eigenschaften und Methoden der Klasse TFileStream

TMemoryStream (unit Classes)	
Eigenschaft	Bedeutung
Position : Int64;	Gegenwärtige Position im Stream
Size: Int64;	Gegenwärtige Größe des Streams
Methode	Bedeutung
function CopyFrom (Source: TStream; Count Int64) : Int64;	Kopiert Count Bytes aus Source in den aktuellen Stream; Funktionswert: tatsächlich übertragene Bytezahl; Count = 0 bedeutet, dass die ganze Datei übertragen wurde. Die Byte- anzahl erhalten Sie dann in Size
Constructor Create;	Erstellt eine neue Instanz von TMemoryString.
Destructor Destroy;	Zerstört den Stream.
Seek (Offset: LongInt; Origin: Word);	Offset: Versatz der Startposition in Bytes Origin: Bezugspunkt für die Suche. Siehe soFromXXX in Ta- belle 20.3.

Tabelle 20.1: Wichtige Eigenschaften und Methoden der Klasse TMemoryStream

21. Modularisierung von Bedienoberflächen

ACHTUNG

Einsatz einer grafischen IDE birgt auch Gefahren.

Planlose Codeerstellung vermeiden! Geeignete Gliederungselemente wie Frames nutzen

Mit einer grafischen IDE wie Lazarus können ansprechende Grafische Bedienoberflächen rasch und intuitiv erstellt werden. Dabei besteht aber die Gefahr, dass der zugehörige Code planlos erstellt wird, rasch anwächst und damit sehr unübersichtlich und sehr schwer wartbar wird. Dem kann durch den Einsatz geeigneter Gliederungselemente wie z. B. Frames abgeholfen werden.

21.1. Das Angebot von Lazarus: Frames als Mittel der Programmgliederung

Die Klasse `TFrame` ermöglicht es, Formulare in einzelne Baugruppen zu gliedern. Dabei können zwei Ziele im Vordergrund stehen:

- Durch Aufgliederung des Formulars in mehrere Frames und die dadurch folgende Verteilung des zugehörigen Programmcodes auf mehrere, dafür aber kleinere und somit leichter überschaubare Dateien kann die Übersichtlichkeit und Wartbarkeit des Programms erheblich verbessert werden.
- Besonders vorteilhaft ist die Verwendung von Frames, wenn mehrere ähnliche visuelle Baugruppen innerhalb eines Formulars zu realisieren sind. Dann wird nur an einer Stelle Code erstellt, der von beliebiger Stelle aus beliebig oft benutzt werden kann.

21.2. Innerhalb ein und desselben Formulars mehrere ähnliche Baugruppen mit Frames realisieren

21.2.1. Aufgabenstellung

Innerhalb eines Formulars sind zwei nebeneinander angeordnete ähnliche Gruppen visueller Komponenten zu realisieren, die jeweils aus einer Schaltfläche und einem Beschriftungsfeld bestehen. Beim Betätigen der linken Schaltfläche wird unter dieser der Text „Hallo Welt“ ausgegeben. Betätigen Sie die rechte Schaltfläche, so erscheint unter dieser der Text „Herzlich willkommen“.

21.2.2. Lösung

Da Sie zwei strukturell gleiche Komponentengruppen benötigen, sind Sie natürlich bestrebt, diese nur einmal zu implementieren. Das erreichen Sie, indem Sie ein Frame erstellen und dieses zweimal im Hauptformular instanziierten (d. h. zweimal auf das Hauptformular ziehen).

Frame einmal konstruieren aber zweimal instanziierten.

21.2.2.1. Erstellen eines Frames

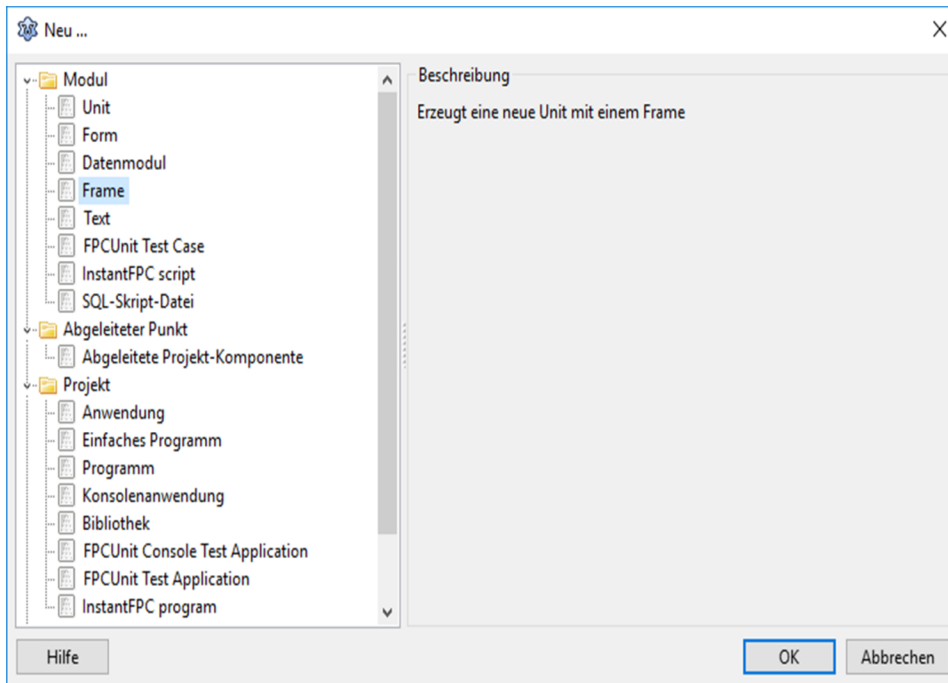


Abbildung 21.1: Formular zur Auswahl eines Lazarus-Bausteinen (eingestellt auf leeren Frame)

Anlegen eines leeren Frames

Öffnen Sie im Zuge der Entwicklung einer Anwendung mit Datei|Neu... das Formular zur Auswahl der Lazarus-Bausteine (Abbildung 21.1). Im linken Teil des Fensters wählen Sie dann die Objektgalerie und wählen Sie dann unter dem Knoten Modul das Element Frame (ggf. scrollen!). Es erscheint eine Struktur ähnlich einem Formular, die Sie jetzt in gleicher Weise wie ein Formular – z. B. indem Sie darauf Steuerelemente platzieren -bearbeiten können. Nach dem Abspeichern wird der Frame im Projekt und in der Werkzeugpalette verfügbar (Abbildung 21.2).

Nur Frames, die in das Projekt aufgenommen wurden, sind mittels der Werkzeugpalette verfügbar.

(Wieder-) Verwendung eines bereits existierenden Frames

Wollen Sie einen bestehenden Frame, der z. B. für ein früheres Projekt entwickelt wurde, nutzen, so müssen Sie diesen zunächst dem Projekt hinzufügen.

Hierfür wird mit Projekt|Projektinspektor der Projektinspektor aufgerufen. Nach Betätigen der Taste  kann im Projekt durch entsprechende Dateiauswahl ein existieren-

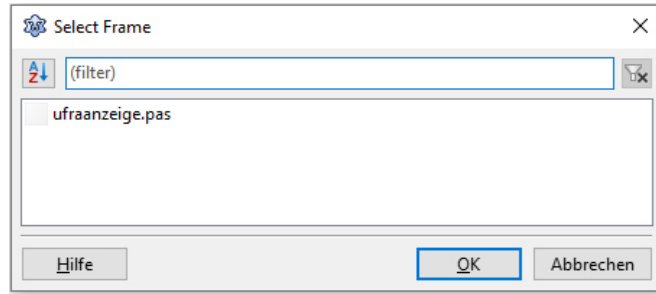


Abbildung 21.2: Dialog zum Auswählen eines Frames

der Frame verfügbar gemacht werden. Mit dem Hinzufügen zum Projekt wird der Frame mittels der Werkzeugpalette verfügbar. Er wird damit aber noch nicht benutzt.

Einbau des Frames in ein Formular

Im nächsten Schritt wird der Frame im Formular platziert. Dazu wird das Frame-Symbol in der Werkzeugpalette angewählt. Daraufhin öffnet sich ein Fenster mit einer Liste der im aktuellen Projekt verfügbaren Frames (Abbildung 21.2). Eine Auswahl in der Liste platziert den gewählten Frame im aktuell in Bearbeitung befindlichen Formular oder Frame.

Bei Verwendung von Frames wird ein Formular hierarchisch gegliedert. Statt ein Formular direkt mit Anzeige- und Steuerelementen zu füllen wird es in einzelne Frames gegliedert, die die Anzeige- und Steuerelemente enthalten und vorab erstellt wurden. In umfangreichen Formularen können Frames auch hierarchisch in mehreren Ebenen eingesetzt werden. D. h. Frames können selbst wieder Frames enthalten.

Ändern Sie einen Frame z. B. indem Sie eine darin befindliche Schaltfläche verschieben, so werden diese Änderungen automatisch an allen Orten des Projekts übernommen, an denen dieser Frame mit den Standardwerten genutzt wird.

Eigenschaftswerte, die bei der Definition einer Frameklasse festgelegt wurden (Standardwerte), können bei der Nutzung des Frames lokal geändert und überschrieben werden. Diese Änderungen werden durch Einträge in der Formulardatei desjenigen Formulars festgehalten, das den Frame instanziert.

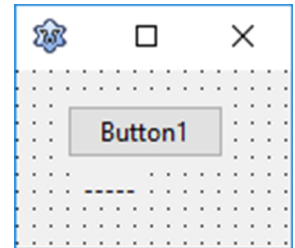


Abbildung 21.3: Innerhalb eines Softwareprojekts mehrfach verwendeter Programmbaustein auf der Basis von TFrame.

Framehierarchie:
Frames können selbst wieder aus Frames bestehen.

22. Eingriffe in den Programmablauf

22.1. Abbruch einer laufenden Berechnung zu einem beliebigen Zeitpunkt

22.1.1. Aufgabenstellung

In der professionellen Programmierung ist es häufig erforderlich, den Abbruch einer Berechnung durch den Bediener vorzusehen. Dabei ist wichtig, dass zwar die Berechnung aber nicht auch das Programm abgebrochen wird.

NUR die Berechnung abbrechen, NICHT das Programm!!

Beispielhaft soll hier ein Programm erstellt werden das nacheinander die Quadrate der Zahlen von 0 bis 30000 berechnet und deren Wert sofort nach der Berechnung anzeigt. Ein Abbruch dieses Programms muss zu jedem beliebigen Zeitpunkt vorgenommen werden können.

22.1.2. Erster Lösungsversuch

Oberflächlich betrachtet erscheint Ihnen die Erstellung dieses Programms wahrscheinlich ganz einfach. Wobei Sie vielleicht folgende Gliederung wählen:

Die Lösung scheint ganz einfach. - Sie ist es aber nicht!!

- Bedienoberfläche
- Berechnungsschleife
- Abbruchmechanismus

22.1.2.1. Bedienoberfläche

Die Bedienoberfläche (s.a. 22.1) besteht aus einem Formular, das zwei Schaltflächen (*BtnStart* und *BtnStop*) und ein Beschriftungsfeld (*LblResultat*) enthält. Die Schaltflächen dienen dem Starten und Stoppen der Berechnung. Im Beschriftungsfeld soll während der Programmlaufs immer das aktuelle Resultat angezeigt werden.

22.1.2.2. Berechnungsschleife

Die Berechnungsschleife wird in der Ereignismethode des *OnClick*-Ereignisses der Schaltfläche *Start* untergebracht. Diese Schleife wird entweder durchlaufen bis der Schleifenzähler den vorgesehenen Maxi-

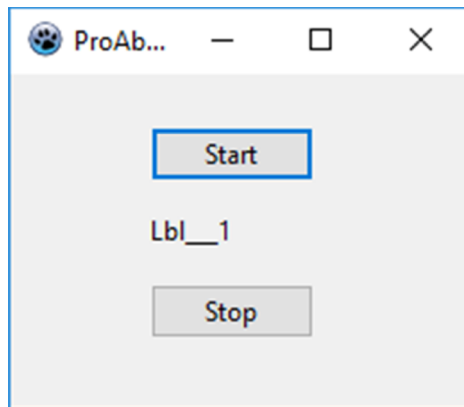


Abbildung 22.1: Im Lösungsversuch verwendete Bedienoberfläche

malwert (*MaxZaehl*) von 30000 erreicht oder bis die boolesche Variable *bEnde* den Wert *true* annimmt.

22.1.2.3. Abbruchmechanismus

Der Abbruchmechanismus basiert auf der Ereignismethode für das *OnClick*-Ereignis der Schaltfläche *BtnStop*. Das Betätigen der Stop-Schaltfläche startet die Ereignismethode. Innerhalb der Ereignismethode wird die Variable *bEnde* auf *true* gesetzt. Der Code für Berechnungsschleife und (vorgesehenen) Abbruchmechanismus ist nachstehend dargestellt:

Implementationsdatei UFrmainAbbruchVorstufe1.pas

```
unit UFrmainAbbruchVorstufe1;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils, FileUtil, Forms, Controls, Graphics, Dialogs, StdCtrls;

type

  { TFrmainAbbruchVorstufe1 }

  TFrmainAbbruchVorstufe1 = class(TForm)
    BtnStart: TButton;
    BtnStop: TButton;
    Lbl__1: TLabel;
    procedure BtnStartClick(Sender: TObject);
    procedure BtnStopClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    bEnde: boolean;
    MaxZaehl: integer;
  end;

var
  FrmainAbbruchVorstufe1: TFrmainAbbruchVorstufe1;

implementation

{$R *.lfm}

{ TFrmainAbbruchVorstufe1 }

procedure TFrmainAbbruchVorstufe1.BtnStartClick(Sender: TObject);
var
  i : integer;
begin
  Lbl__1.Caption := '';
```

```

bEnde := false;
for i := 0 to MaxZaehl do
begin
  Lbl__1.Caption := IntToStr (i*i);
  if bEnde then ///Abbruch falls BtnStop betätigt wurde.
    break;
end;

end;

procedure TFrmMainAbbruchVorstufel.BtnStopClick(Sender: TObject);
begin
  bEnde := true;
end;

procedure TFrmMainAbbruchVorstufel.FormCreate(Sender: TObject);
begin
  MaxZaehl := 30000;
end;

end.

```

22.1.2.4. Testergebnis

Wenn Sie das Programm starten, erkennen Sie, dass der Text im Beschriftungsfenster zunächst unverändert bleibt und dann nach kurzer Zeit den Wert 900.000.000 annimmt. Dieses Programmverhalten ist unabhängig davon, ob die Stop-Schaltfläche betätigt wird oder nicht!

Weshalb wirkt die Stop-Schaltfläche nicht?

Was passiert hier?

Wenn die Berechnung wie oben dargestellt programmiert wird, d. h. ohne besondere Maßnahmen, dann wird sie nach Start der Berechnung ohne Unterbrechung abgearbeitet. Der Prozessor ist während dieser Zeit voll ausgelastet. Die Veränderung der Anzeige oder die Ausführung von *FormMainStopClick* können erst erfolgen, wenn die Methode *FormMainStartClick* beendet wurde.

22.1.3. Lösung

Mit ein paar kleinen Veränderungen können Sie die in 22.1.2.4 geschilderten Probleme rasch beheben.

22.1.3.1. Schritthaltende Resultatsanzeige

Eine schritthaltende Resultatsanzeige erhalten Sie unabhängig von anderen Maßnahmen indem Sie per Programm veranlassen, dass das Beschriftungsfeld *LblResultat* mittels der Methode *Repaint* nach jeder Berechnung neu dargestellt wird. Diese Maßnahme führt allerdings zu einer erheblichen Laufzeitverlängerung.

23. Anspruchsvolle Geschäftsgrafiken problemlos erstellen (Einsatz des TACHart-Rahmenwerks)

In Lazarus ist mit TACHart ein Rahmenwerk (Framework) zur Erstellung von Geschäftsgrafiken enthalten. Es ähnelt dem TeeChart-Rahmenwerk¹⁷, das Bestandteil vieler C++-Builder oder Delphi-Versionen war. TACHart ist vollkommen in die Lazarus-Entwicklungsumgebung integriert. Die Komponenten von TACHart finden Sie auf der Werkzeugpalette unter dem Reiter Chart.

Ein Beispiel für eine solche Komponente ist TChart, eine universelle Komponente für die grafische Darstellung von Datenmengen. TChart ist ein Bestandteil des TACHart-Rahmenwerks.

Aufgrund des Umfanges und der Komplexität der TACHart-Software und ihrer dadurch gegebenen fast unüberschaubaren Gestaltungsmöglichkeiten kann hier nur eine kurze Einführung in die Nutzung von TACHart gegeben werden. Es geht dabei ausschließlich darum, das Prinzip der Anwendung und des Aufbaus aufzuzeigen. Darauf aufbauend können Sie dann Details für Ihre spezielle Nutzung selbst erarbeiten.

Leider sind die Komponenten und Funktionen von TACHart nur vergleichsweise schwach dokumentiert. Für die vertiefte Einarbeitung bieten sich folgende Möglichkeiten:

Das TACHart-Rahmenwerk ist vergleichsweise schwach dokumentiert!

- **Verschiedene Dokumente** (z. B. eine Basisinformation mit Dokumentationsübersicht [FPWTACHA], eine einführende Dokumentation [FPWTACHD] und ein Tutorial [LFPTCTUT]) im Internet, vor allem im Freepascal- und Lazarus-Wiki. [FPWTACHA] enthält eine umfangreiche Linkliste auf weitere Dokumente zum Thema TACHart.
- **Die Dokumentation von TeeChart.** TeeChart weicht zwar von TACHart ab. An vielen Stellen liefert die Dokumentation von TeeChart [STEEMA0] jedoch gute Anregungen zum Verständnis von TACHart und gestaltet auf diese Weise das immer wieder notwendige Experimentieren effektiver. Lt. Aussage der TACHart-Autoren streben sie keine Kompatibilität zu TeeChart an. Trotzdem kann nach Erfahrung des Autors ein Blick in die Unterlagen von TeeChart nach kritischer Prüfung in vielen Fällen weiterhelfen.
- **Eine umfangreiche Beispielsammlung**, die in [LFPTCDEM] beschrieben ist und mit Lazarus mitgeliefert wird.

Beispiele für interaktive und textorientierte Programmierung

¹⁷ TeeChart ist ein Rahmenwerk für Geschäftsgrafiken, das in den meisten neueren Versionen von Delphi enthalten ist. Hersteller ist die Firma Steema [STEEMA0].

- Die Quellen von TACHart, die mit Lazarus ebenfalls mitgeliefert werden. Zu finden in: **Installationsverzeichnis_von_Lazarus\components\TACHart**

Im folgenden wird ein Beispiel gründlich erarbeitet. Die Darstellung der dabei benutzten Softwarekomponenten muss sich dabei in der Regel auf die im Beispiel verwendeten Eigenschaften und Methoden beschränken.

Jahr	2002		2005		2009	
Anteil Zweitstimmen/Sitze	Stimmen %	Sitze	Stimmen %	Sitze	Stimmen %	Sitze
CDU/CSU	38,5	248	35,2	226	33,8	239
SPD	38,5	251	34,2	222	23	146
FDP	7,4	47	9,8	61	14,6	93
Grüne	8,6	55	8,1	51	10,7	68
Linke	4	2	8,7	54	11,9	76
Sonstige	4	-	4	-	6	-

Tabelle 23.1: Ergebnisse der Bundestagswahlen von 2002 bis 2009

Quelle: <http://www.wahlrecht.de/ergebnisse/bundestag.htm>

Jahr	1998		2003		2008	
Anteil Zweitstimmen/Sitze	Stimmen %	Sitze	Stimmen %	Sitze	Stimmen %	Sitze
CSU	54,1	123	62	124	44,2	92
SPD	28,1	67	19,2	41	18,1	39
FW	3,2	-	3,6	-	9,8	21
Grüne	5,9	14	7,7	15	9,8	19
FDP	1,6	-	2,5	-	7,8	16
Sonstige	7,2	-	5	-	10,3	-

Tabelle 23.2: Ergebnisse der Bayrischen Landtagswahlen von 1998 bis 2008

Quelle: <http://www.statistik.bayern.de/wahlen/landtagswahlen/>

23.1. Aufgabenstellung

23.1.1. Fachaufgabe

Erstellen Sie ein Programm, das die Ergebnisse der Bundestagswahlen der Jahre 2002, 2005 und 2009 und der Bayrischen Landtagswahlen der Jahre 1998, 2003 und 2008 darstellt. Darzustellen sind die Ergebnisse der fünf stärksten Parteien. Die Ergebnisse der übrigen Parteien werden unter Sonstige zusammengefasst. Weiterhin ist für alle Fälle die Ergebnisveränderung gegenüber der vorhergegangenen Wahl darzustellen.

Erstellen Sie sowohl Diagramme für die Darstellung der Sitzverteilung als auch für die Darstellung der erreichten prozentualen Anteile. Die Anzeige der Sitzverteilung, der prozentualen Ergebnisse und der Ergebnisveränderungen (aktuelles Ergebnis – letztes Ergebnis) soll in Form von Liniendiagrammen, die Ergebnisanzeige (Sitze und prozentual) alternativ auch als Balkendiagramm, Kreisdiagramm (Tortendiagramm) oder Blasenendiagramm erfolgen.

Für die Ergebnisse der Bundestagswahlen ist eine 2D-Darstellung, für die der Landtagswahlen wahlweise auch eine 3D-Darstellung vorzusehen.

23.1.2. Aufbau und Vorgehen

Stellen Sie auf einem Formular in einem Diagramm die Wahlergebnisse der Bundestagswahlen und auf einem zweiten die der Bayrischen Landtagswahlen dar. Ein zusätzliches Hauptformular soll ausschließlich die Auswahl zwischen den Ergebnissen der Bundestagswahlen und der Bayrischen Landtagswahlen ermöglichen.

In den behandelten Beispielen zu TChart wird die Darstellung der Bundestagswahlergebnisse weitgehend interaktiv programmiert (also durch „Anklicken“ oder „Ziehen und Ablegen“) während die Darstellung der Landtagswahlergebnisse klassisch codiert wird.

23.2. Lösung mit dem TChart-Rahmenwerk

Bevor Sie in die Programmierung der Geschäftsgrafiken einsteigen, sollten Sie sich unbedingt mit den Möglichkeiten der Klasse TChart und weiterer Klassen aus deren Umfeld befassen. Drei Klassen (bzw. deren Nachkommen) kommt eine besondere Bedeutung zu:

- Der Diagrammklasse TChart, die für die Darstellung des gesamten Diagramms verwendet wird.
- Den von TBasicChartSeries (Abbildung 23.3, Seite 188) abgeleiteten Klassen, die für das optische Erscheinungsbild einzelner Datenmengen im Diagramm stehen.

24. Nutzen Sie die Möglichkeiten Dynamischer Link-Bibliotheken (DLLs)

24.1. Ein bisschen Theorie

Eine DLL (Dynamic Link Library, Dynamische Link Bibliothek) ist eine Bibliothek, die dynamisch an das jeweilige Hauptmodul (z. B. ein Hauptprogramm) angebunden werden kann. Sie wird ihrem Nutzer also erst zur Laufzeit zugeordnet.

Man unterscheidet - und das klingt fast ein bisschen paradox - die implizite (statische) Anbindung und die explizite (dynamische) Anbindung. Die implizite Anbindung erfolgt zu Beginn des Prozesses, also zur Beginn der Laufzeit. In diesem Fall muss die benötigte DLL bereits zu Beginn der Laufzeit vorhanden sein. Der Arbeitsspeicherplatz zur Aufnahme der DLL muss während der gesamten Dauer des Prozesses bereit gestellt werden.

Wird die DLL explizit angebunden, so kann dies vollkommen bedarfsorientiert geschehen. Die DLL wird dann ausschließlich für den Zeitraum, in dem sie benötigt wird angebunden, unmittelbar danach kann sie wieder abgetrennt werden. Der Ressourcenbedarf für die DLL muss also nur in dem Zeitraum befriedigt werden, wo er tatsächlich auch vorhanden ist. Mehrere nicht zeitgleich benötigte DLLs können sich denselben Speicherplatz teilen.

24.1.1. Möglichkeiten der DLL

DLLs sind ein wesentliches Element des professionellen Softwareengineering. Sie bieten die folgenden Vorteile:

- Es gibt ein extrem breites Angebot vorgefertigter professioneller Lösungen.
- Da die Bibliotheksmodule erst zur Laufzeit gebunden werden, ist die EXE-Datei selbst meist wesentlich kleiner als bei Verwendung herkömmlicher statischer Bi-

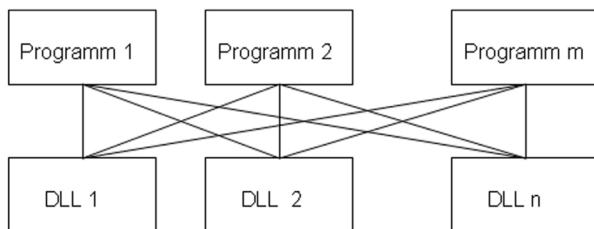


Abbildung 24.1: Eine DLL kann von mehreren Programmen genutzt werden.

bibliotheken. Für den gesamten Speicherplatzbedarf des jeweiligen Programms trifft das hingegen nicht zu.

- Das Hauptprogramm (EXE-Datei) und die einzelnen DLLs können in unterschiedlichen Programmiersprachen erstellt werden.
- Wartungsarbeiten reduzieren sich auf einfaches Kopieren und ggf. Überschreiben an Stelle von Binden und Installieren.

Zusammenstellung des Laufzeitsystems erfordert mehr Aufwand und Sorgfalt.

Nachteilig ist, dass die Zusammenstellung des Laufzeitsystems etwas mehr Aufwand und vielleicht auch Sorgfalt erfordert. Einmal richtig zu binden reicht in diesem Fall meist nicht aus.

24.1.1.1. Nutzung einer DLL durch mehrere Programme

Wenn eine Aufgabe von mehreren Programmen auszuführen ist, dann kann diese Aufgabe von ein und derselben DLL erledigt werden (Abbildung 24.5). Das kann den Umfang der Gesamtheit der EXE-Dateien extrem reduzieren. Üblich ist dies z. B. bei der Implementation von Betriebssystem-Funktionen.

24.1.1.2. Realisierung hybrider Programmkonzepte

DLLs können in den unterschiedlichsten Programmiersprachen erstellt werden. Bekanntlich ist es so, dass für eine gegebene Aufgabenstellung nicht jede Programmiersprache gleich gut geeignet ist. Verwendet man für grafische Bedienoberflächen oder arithmetische Berechnungen vorteilhaft C++, Pascal oder auch Java, so wird man für die Lösung logischer Aufgabenstellungen eher Prolog verwenden. Mit dem DLL-Konzept kann jeder Programmteil in der für ihn angemessenen Programmiersprache erstellt werden. Hiervon ist eine deutliche Rationalisierung der Programmentwicklung zu erwarten.

Im Abschnitt 24.5 wird gezeigt, wie C++-DLLs durch Lazarus-Programme genutzt werden können. Die weitere Sprachkombinationen sollen in einem Folgeband behandelt werden (Planung).

24.1.1.3. Skalierung von Programmen

Unterschiedliche Programmversionen können durch unterschiedliche DLLs bzw. unterschiedliche Ausführungen gleichnamiger DLLs, bzw. gleichnamiger Module in DLLs realisiert werden. Dabei werden einfach die erforderlichen Dateien an einem definierten Ort zusammenkopiert. Ein Bindevorgang (Linkvorgang) ist nicht erforderlich. So können durch einfaches Zusammenstellen von Dateien verschiedene Programmversionen (z. B. Light, Standard, Professional, ...) generiert werden.

24.1.1.4. Verbesserung der Wartbarkeit

Im Gegensatz zur Verwendung statischer Bibliotheken, kann bei Verwendung von DLLs das Programm dadurch geändert werden, dass einzelne DLLs ausgetauscht

werden. Ein Binde- (Link-) Vorgang, der in der Regel nur vom Entwickler durchgeführt werden kann, entfällt damit und die EXE-Datei bleibt unverändert. Dieser Wartungsvorgang kann in verschiedenen Graden automatisiert werden. Das reicht von der Information des Benutzers über Neuerungen mit anschließendem manuellem Dateiersatz bis zum vollautomatischen Ersatz veralteter DLLs durch das Programm selbst (z. B. mittels eines Vorschaltprogramms oder beim Programmstart). Ersatz beim Programmstart setzt voraus, dass die DLLs explizit angebunden werden (24.2.8).

24.2. So erstellen Sie eine DLL

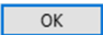
Im folgenden wird die DLL-Technologie anhand eines einfachen Anwendungsbeispiels dargestellt. Es wird eine DLL entwickelt, die eine Funktion und eine Variable bereitstellt.

24.2.1. Aufgabenstellung

In der DLL soll die Funktion $\sin(a \cdot x)$ implementiert werden. Die Schnittstelle beinhaltet die Variable *amult* vom Typ *double* mittels derer der Faktor *a* vorgegeben wird sowie die Funktion *sina* (*x*: *double*): *double*, die den Wert $\sin(a \cdot x)$ liefert. *sina* basiert auf *sin*. Der Zusammenhang lautet: $\text{sina} = \sin(a \cdot x)$.

24.2.2. Die Lösung

24.2.2.1. Die DLL – ein Lazarus-Projekt

Die Erstellung einer DLL beginnt fast wie die eines Programms. Wählen Sie Datei|Neu... Es öffnet sich ein Formular. Wenn Sie dort im linken Fenster im Ordner Projekt auf Bibliothek klicken, nimmt es das in Abbildung 24.2 (Seite 238) gezeigte Bild an. Nach dem Klicken auf die Schaltfläche  erscheint ein leeres Bibliotheksprojekt, das den nachstehenden Code umfasst:

```
library Project1;

{$mode objfpc}{$H+}

uses
  Classes
  { you can add units after this };

begin
end.
```

Dieses leere Projekt kann z. B. mittels der Aktionsfolge Start|Kompilieren übersetzt werden. (Alternativen wären auch Start|Neu Kompilieren und Start|Schnelles Kompilieren). Ein Übersetzen unter Verwendung der Schaltfläche Start (F9) bzw. dem

25. Indy öffnet die Tür zum Internet

25.1. Übersicht über die Aufgabenstellungen

In diesem Kapitel sollen beispielhaft drei typische Internetapplikationen erstellt werden:

- Ein Postausgangsklient, das ist ein Programm oder ein Programmteil, das von einem verteilten Rechner die Post (Email) versendet.
- Ein Posteingangsklient, das ist ein Programm oder ein Programmteil, das auf einem verteilten Rechner die Post (Email) entgegennimmt.
- Ein Datenübertragungs- (FTP-) Client, das ist ein Programm oder ein Programmteil mittels dessen Hilfe beliebige Dateien auf einen zentralen Rechner geladen oder von diesem abgerufen werden.

Komponenten, die die Lösung dieser Aufgaben unterstützen, sind nicht originärer Bestandteil von Lazarus. Eine probate Möglichkeit ist die Verwendung geeigneter Bausteine, die von Drittanbietern angeboten werden. Besonders vorgeprüfte Komponenten von Drittanbietern werden Ihnen direkt in der integrierten Entwicklungsumgebung von Lazarus angeboten (s. s. 25.2.2). Das gilt auch für Indy, ein universales Rahmenwerk für internetbezogene Anwendungen.

25.2. Was bietet mir Lazarus - Die Indy-Bibliothek

25.2.1. Geschichte und Inhalt

Indy²⁵ (Internet Direct, früher Winshoes) ist ein Open Source Projekt das von Freiwilligen mit Hilfe industrieller Sponsoren bearbeitet wird [INDY]. Seit den ersten Anfängen von Chad Z. Hower im Jahre 1993 hat sich Indy stark entwickelt und ist heute eine umfassende Socket-Bibliothek, die auf verschiedenen Plattform verfügbar ist. Anwendungsprogrammierern stellt Indy einen einfachen und zuverlässigen Zugriff auf die Dienste des Internets zur Verfügung zu stellen. Mehr als 100 Hochebenen-Protokolle sind leicht zugänglich implementiert, darunter SMTP, POP3, HTTP und NNTP.

Schon 2001 hat Borland Indy im Rahmen seiner Softwareentwicklungsumgebungen C++Builder und Delphi lizenziert. Darüber hinaus werden Kylix, C#, Visual Basic .NET und Delphi .NET unterstützt.

25 Der Einfachheit halber wird hier allein das Wort Indy sowohl als Bezeichnung für das Projekt wie auch des Rahmenwerks verwendet.

Das Indy-Handbuch umfasst mehr als 4500 Seiten! Bisher ist es nur für Delphi verfügbar.

Das Indy-Rahmenwerk implementiert ca. 600 Klassen. Das Handbuch umfasst mehr als 4500 Seiten. Es liegt derzeit ebenso wie auch die Hilfeseiten leider nur für Delphi vor. Die Vorgehen für Lazarus muss aus den existierenden Unterlagen durch Analogschluss abgeleitet werden. Die Klassen von Indy sind meist sehr umfangreich. So besitzt die Klasse `TIdSMTP`, die beim Postausgangs-Client zum Einsatz kommt, 32 Methoden. Bei den anderen verwendeten Klassen sind die Verhältnisse ähnlich.



Abbildung 25.1: Hilfe-Seite mit Informationen zu Klassenhierarchie und (Header-) Datei

Nur Lösungsprinzip, keine Details! Keine Perfektionierung!

Schon aus diesen Zahlen geht hervor, dass weder das gesamte Indy-Projekt noch einzelne wichtige Klassen desselben im Rahmen dieses Buches erschöpfend behandelt werden können. Dementsprechend können die Beispiele auch nur das Prinzip nicht aber universelle Lösungen aufzeigen.

Wer tiefer in Indy einsteigen will oder muss, dem bleibt das Studium der o. e. Dokumentation nicht erspart. Hierfür gilt folgende Empfehlung:

Schnittstellenbestimmung über die Delphi-Dokumentation

Die grundsätzlichen Details zu Indy finden Sie in der (Delphi-) Dokumentation und in der Online-Hilfe. Die dortigen Angaben gelten sinngemäß auch für die Programmierung in Lazarus. Für die Methode `Get` der Klasse `TIdFTP` (FTP-Client) finden Sie z. B. in der Delphi-Dokumentation die folgende Schnittstelle:

```
procedure Get(
  const ASourceFile: string;
  const ADestFile: string;
  const ACanOverwrite: boolean = false;
  AResume: boolean = false
```

```
); overload;
```

Diese Schnittstelle gilt 1:1 auch für Lazarus

25.2.2. Laden und Installieren von Paketen mit dem Paket-Manager

Ab der Version 1.8. von Lazarus wird der Paket-Manager mitgeliefert. Er stellt selbst ein Paket dar, das mitgeliefert wird aber nicht installiert ist.

- Package | *Installierte Packages einrichten ...* anwählen.
- Unter *Verfügbar für Installation* *onlinepackagemanager 1.0* anwählen.
- Den Paket-Manager durch Betätigen der Schaltfläche Auswahl installieren installieren.
- Die Schaltfläche Speichern und IDE rekompilieren betätigen. Die IDE wird neu kompiliert und gestartet.

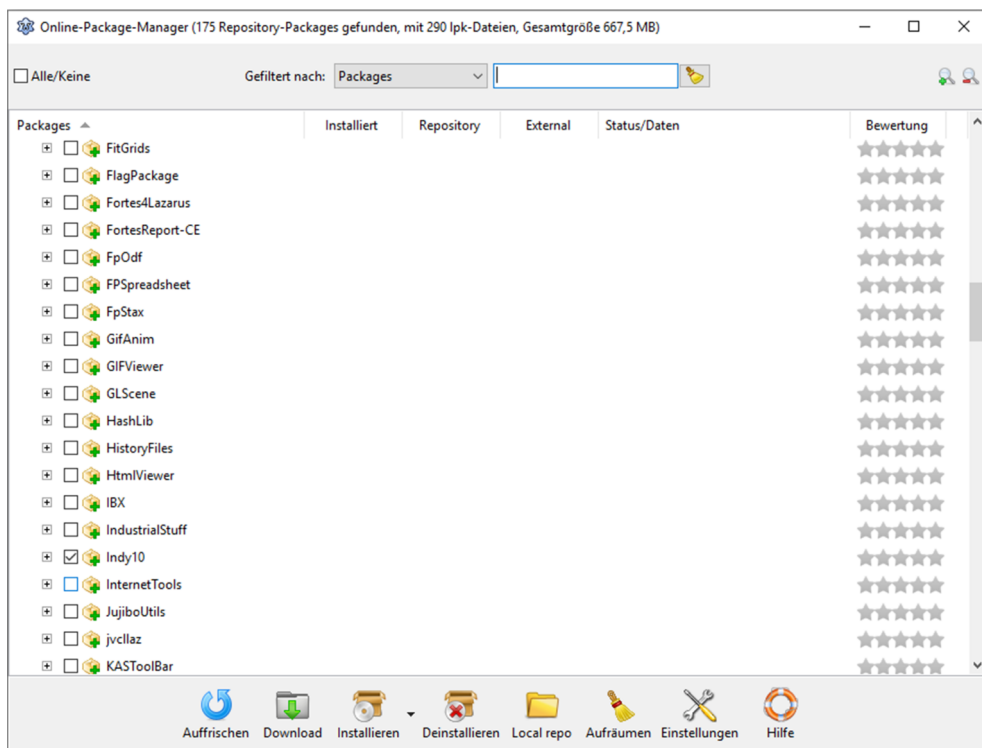


Abbildung 25.2: Oberfläche des Online-Package-Managers

26. Am Ende ist nicht alles vorbei (Teil 2) - Fortgeschrittene Persistenzlösungen / Arbeiten mit Datenbanken

Dieses Kapitel eröffnet Ihnen den Zugang zur schnellen Erstellung einfacher Datenbankanwendungen. Es geht dabei um einen niederschweligen Zugang zu den Kernaufgaben der Anwendungserstellung. Auf eine Fehlerbehandlung wird in den Beispielen aus Gründen der Übersichtlichkeit weitgehend verzichtet.

Lazarus bietet zahlreiche Komponenten zur Unterstützung unterschiedlicher Datenbanktechnologien

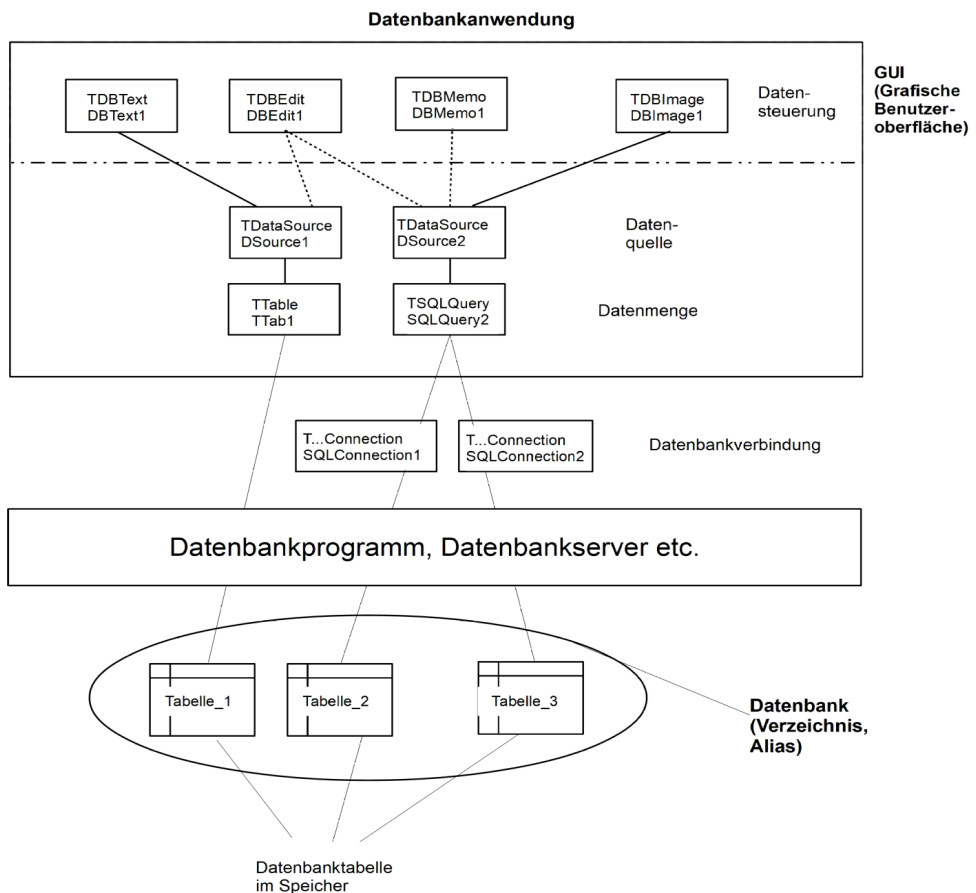


Abbildung 26.1: Aufbau einer Lazarus-Datenbank-Anwendung

- Einfachdatenbanken im Arbeitsspeicherbereich des Lazarus-Programms mit fester (TFixedFormatDataset) oder variabler (TSDFDataset, SDF = Server Data Files) Weite der Tabellenfelder.
- Einfachdatenbanken im Arbeitsspeicherbereich des Lazarus-Programms die auf TMemDataSet und TBufDataSet basieren.
- dBase / FoxPro
- Client-Sever-Datenbanken verschiedenster Art
 - Interbase / Firebird
 - MySQL/MariaDB
 -

Interbase ist eine spezielle Datenbankmaschine, die ursprünglich von Borland entwickelt wurde und mit dem Embarcadero RAD Studio mitgeliefert wird. Die kostenlose Version von Interbase hört auf den Namen Firebird. Für alle aufgeführten Technologien gilt, dass der Zugriff auf die Datenbanken aus Lazarus-Builder-Programmen heraus über einheitliche Datenzugriffs- und Datensteuerungskomponenten erfolgt.

MySQL/MariaDB-Datenbanken sind komfortabel, vielseitig und leicht zu realisieren. Aus diesem Grund wird hier eine umfangreiche Datenbankanwendung einschließlich der Berichtserstellung zunächst mit MariaDB entwickelt. Eine Umstellung dieser Lösung auf Interbase/Firefox ist ggf. leicht möglich.

Am Anschluss an das Thema SQL-Datenbanken wird kurz angedeutet, wie Datenbank-anwendungen mit In-Speicher-Lösungen entwickelt werden können.

Abgeschlossen wird das Kapitel mit einer Einführung in die Erstellung von Datenbank-berichten

26.1. Die Architektur von Datenbank Anwendungen

Auch wenn das Spektrum der mit Bordmitteln²⁹ von Lazarus verwendbaren Datenbanken sehr breit und vielleicht sogar etwas verwirrend ist, so beinhaltet doch jede Anwendung doch gewisse Standardbausteine, die die Standardaufgaben der Datenbankbehandlung lösen:

- Benutzerschnittstelle (GUI)
- Datenquelle
- Datenmenge

²⁹ Der Begriff Bordmittel wird in diesem Buch für solche Komponenten, Bibliotheken, Rahmenwerke usw. verwendet, die mit FreePascal oder der Lazarus-Entwicklungsumgebung mitgeliefert werden.

- Datenbankverbindung
- Datensatz

Diese Standardbausteine werden vom C++Builder in großer Zahl und vielen Varianten zur Verfügung gestellt, sodass die Erstellung von Datenbankanwendungen in vielen Fällen mit vergleichsweise geringem Codieraufwand möglich ist.

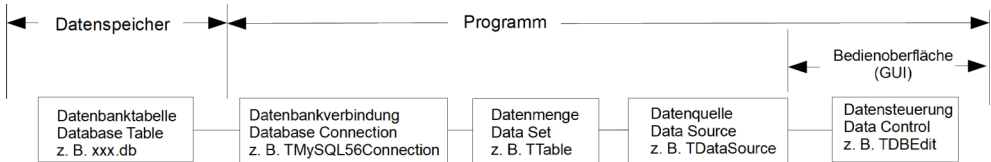


Abbildung 26.2: Datenbankimplementation in Lazarus. Wirkungskette von der Datenbanktabelle bis zur Bedienoberfläche

26.2. Der Beitrag von FreePascal und Lazarus

Das mit obiger Auflistung beschriebene Architekturkonzept wird von Lazarus gut unterstützt. FreePascal und Lazarus bieten zahlreiche Programmbausteine, die die Datenbankprogrammierung ermöglichen bzw. erleichtern. In der Dokumentation zu Lazarus finden Sie eine ganze Menge grundlegender Informationen z. B. in [LFPDABAS] und [LAZARTDB].

Die entsprechenden visuellen Komponenten finden Sie in der Werkzeugpalette standardmäßig auf den Karten Data Controls, DataAccess und SQLdb. Komponenten für die Berichtserstellung, hierunter fällt z. B. das Schreiben der Rechnung), finden Sie unter LazReport.

26.2.1. Benutzerschnittstelle

Die Benutzerschnittstelle der Datenbanken ähnelt weitgehend der bis jetzt behandelten Schnittstelle, die die Formularanwendungen verwenden.

Für die wichtigsten Dialogelemente vom Typ `Txyz` gibt es auf der Karte DataControls ein optisch gleich erscheinendes, datensensitives Steuerelement vom Typ `TDBxyz` (z. B. gibt es `TDBLabel` entsprechend `TLabel`). Datensensitiv bedeutet, dass es eine Schnittstelle zur Datenbank besitzt, sodass Daten praktisch ohne Programmieraufwand in beide Richtungen zwischen Dialogelement und Datenbank übertragen werden können.

Die Elemente für die Benutzerschnittstelle finden Sie unter DataControls.

Inhalt:

In diesem Band werden u. a. die folgenden Themen behandelt:

- Installation der Entwicklungsumgebung Lazarus
- Grundlagen der Programmierung in (Free-) Pascal
- Modulare Programmierung mit Units
- Erstellung einfacher grafischer Bedienoberflächen
- Nutzung des Druckers aus eigenen Programmen
- Erstellung und Bearbeitung einfacher Grafiken
- Visualisierung dynamischer Abläufe

Durch zahlreiche charakteristische Anwendungsbeispiele wird der Leser rasch in die Lage versetzt, individuelle Anwendungen mit Lazarus selbst zu erstellen. Selbstverständlich wird dabei auf typische Fallstricke deutlich hingewiesen.

Weitere Bände, die sich u. a. mit Datenbanktechniken, Internetanwendungen und software-technologischen Aspekten befassen. Weitere Informationen zum Buch finden Sie unter www.informatik-ganz-einfach.de.

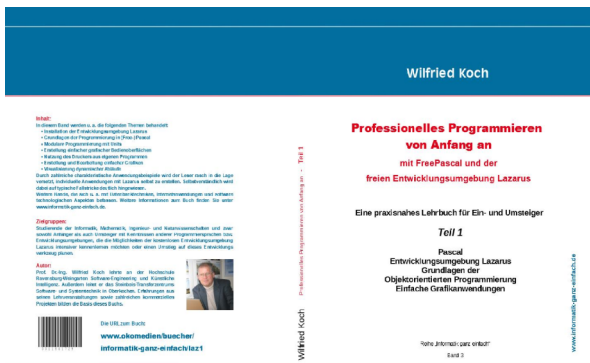
Zielgruppen:

Studierende der Informatik, Mathematik, Ingenieur- und Naturwissenschaften sowie Anwendungsprogrammierer mit Grundkenntnissen in C++, die die Möglichkeit des von Lazarus intensiver kennenlernen möchten oder einen Umstieg auf dieses Entwicklungswerkzeug planen.

Erhältlich beim Oberkochener Medienverlag, booklooker.de oder im Buchhandel. Die Preisangaben gelten nur für Deutschland.

Der Versand durch den Verlag erfolgt innerhalb Deutschlands kostenfrei.

www.okomedien.de; Oberkochener Medienverlag, Dopplerweg 3, 73447 Oberkochen



Buch (Druck) 452 Seiten, Paperback, Hochglanz, 17*22cm, 45 Abbildungen	D: 26,99 €	ISBN 978-3-945899-01-4
Buch mit CD (enthält die Programme des Buchs)	D: 32,99 €	ISBN 978-3-945899-08-3
CD zu Teil 1	D: 11,99€	ISBN 978-3-945899-07-6

Preisliche und technische Änderungen vorbehalten!

Inhalt:

In diesem zweiten Band werden u. a. die folgenden Themen behandelt:

- Eingriffe in den Programmablauf
- Erstellung von Geschäftsgrafiken mit der Komponente TChart
- Einführung in die Programmierung relationaler Datenbanken
- Erstellung von Datenbank-Berichten mit Rave Reports
- Dynamische Link-Bibliotheken
- Ankopplung von Moduln aus anderen Programmiersprachen
- Verbindung zum Internet mit den INDY-Komponenten

Durch zahlreiche charakteristische Anwendungsbeispiele wird der Leser rasch in die Lage versetzt, individuelle Anwendungen mit dem C++Builder selbst zu erstellen. Selbstverständlich wird dabei auf typische Fallstricke deutlich hingewiesen.

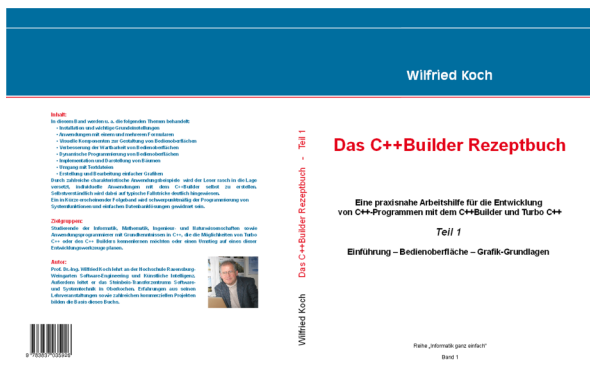
Weitere Bände, die sich u. a. mit fortgeschrittenen Datenbanktechniken, Data Snap – Client/Server-Lösungen und der Nutzung in integrierter Softwareentwicklung befassen sind geplant. Weitere Informationen zum Buch finden Sie unter www.informatik-ganz-einfach.de

Zielgruppen:

Studierende der Informatik, Mathematik, Ingenieur- und Naturwissenschaften sowie Anwendungsprogrammierer mit Grundkenntnissen in C++, die die Möglichkeiten des C++ Builders intensiver kennen lernen möchten oder einen Umstieg auf dieses Entwicklungswerkzeug planen.

Erhältlich beim Oberkochener Medienverlag, booklooker.de oder im Buchhandel. Die Preisangaben gelten nur für Deutschland.

Der Versand durch den Verlag erfolgt innerhalb Deutschlands kostenfrei.



www.okomedien.de; Oberkochener Medienverlag, Dopplerweg 3, 73447 Oberkochen

Buch mit CD (enthält die Programme des Buchs)	D: 20,99 €	ISBN 978-3-849599-11-3
CD zu Teil 2	D: 10,99 €	ISBN 978-3-849599-17-5
CD zu Teil 1 und 2	D: 13,99 €	ISBN 978-3-849599-00-7

Preisliche und technische Änderungen vorbehalten!

Softwareentwicklung mit Lazarus

Vorteile fast ohne Grenzen

- Keine Lizenzgebühren
- Schnelle Entwicklung durch RAD-Umgebung
- Geringe Laufzeiten dank Kompilierung
- Quelloffen
- Hoher Dokumentationswert

Für Ihre Projekte bieten wir Ihnen die maßgeschneiderte Unterstützung

- Beratung
- Konzeption
- Training
- Coaching
- Programmierung

Steinbeis-Transferzentrum Software- und Systemtechnik

Dopplerweg 3; 73447 Oberkochen

Tel: 07364-5335

Mail: info@stz-sosy.com

URL: www.stz-sosy.com