

RAD Studio (Allgemein)

Inhalt

Alle Anleitungen	1
Anwenden der aktiven Build-Konfiguration auf ein Projekt	2
Packages erstellen	3
Compilieren von Entwurfszeit-Packages, die Delphi-Quelltextdateien enthalten	4
Erstellen von Build-Ereignissen	5
Erstellen von benannten Build-Konfigurationen für C++	6
Erstellen von benannten Build-Konfigurationen für Delphi	7
Ressourcendateien im Translation-Manager bearbeiten	8
Weitere Programmiersprachen installieren	9
Delphi-Units in eine Anwendung linken	10
Ein Symbol umbenennen	11
Erzeugen eines Projekts mit einem MSBuild-Befehl	12
Verwenden von Targets-Dateien	14
Arbeiten mit benannten Optionsgruppen	15
Ausdruck hinzufügen	17
Mit einem ausgeführten Prozess verbinden	18
Quelltexthaltepunkte setzen und bearbeiten	19
VCL für .NET-Quelltext debuggen	22
Kurzhinweise während des Debuggens verwenden	23
Werte von Datenelementen untersuchen und ändern	24
Variablenausdrücke bearbeiten	26
Das Projekt zum Debuggen vorbereiten	27
Installieren, Starten und Anhalten des externen Debug-Servers	28
Debugger auf einem externen Computer installieren	30
Übersicht zum Debuggen externer Anwendungen	32
Verbindung für das externe Debuggen einrichten	33

Dateien für das externe Debuggen vorbereiten	35
Die Suchreihenfolge für Debug-Symboltabellen festlegen	36
Verwenden der CPU-Ansicht	38
Erweiterte Informationen zu überwachten Ausdrücken anzeigen	39
Deployment von ASP.NET-Anwendungen	40
Deployment von Anwendungen	41
Code-Folding verwenden	42
Quelltext-Templates erstellen	43
Quelltext-Editor anpassen	44
Tastatur-Makro aufzeichnen	45
Positionsmarken verwenden	46
Klassenvervollständigung verwenden	47
Code Insight verwenden	48
Quelltext-Templates verwenden	50
Mit der Versionsverwaltung arbeiten	52
Sync-Bearbeitungsmodus verwenden	54
Komponenten in ein Formular einfügen	55
Referenzen hinzufügen	56
Dateien hinzufügen und entfernen	57
Templates in die Objektablage einfügen	58
Referenzen in lokalen Pfad kopieren	59
Komponenten-Templates erzeugen	60
Ein Projekt erstellen	61
Formular anpassen	62
Symbolleisten anpassen	63
Tool-Palette anpassen	64
Deaktivieren von Themes in der IDE und in Ihrer Anwendung	65
Tool-Fenster andocken	66

Elemente der Tool-Palette suchen	67
Metadaten von .NET-Assemblierungen untersuchen	68
Windows-Typbibliotheken untersuchen	69
Benutzerdefinierte Komponenten installieren	70
Dateien in der Projektverwaltung umbenennen	71
Desktop-Layouts speichern	72
Komponenteneigenschaften festlegen	73
Dynamische Eigenschaften festlegen	74
Projektoptionen einstellen	75
Festlegen von C++-Standardprojektoptionen	76
Eigenschaften und Ereignisse festlegen	77
Die IDE für das Simulieren von Delphi 7 einrichten	78
Voreinstellungen für Tools festlegen	79
Designer-Richtlinien für VCL-Komponenten verwenden	80
Verwenden des Datei-Browsers	81
To-Do-Listen verwenden	82
Verwenden von virtuellen Ordnern	83
Ereignisbehandlungsroutine schreiben	85
Sprachen zu einem Projekt hinzufügen	86
Ressourcendateien im Translation-Manager bearbeiten	88
Aktive Sprache für ein Projekt festlegen	90
Externen Translation-Manager einrichten	91
Ressourcenmodule aktualisieren	93
Externen Translation-Manager verwenden	94
Konfigurieren des Speichermanagers	95
Vergrößern des Speicheradressraums	97
Speicherverwaltung	98
Überwachen der Speicherverwendung	99

Registrieren von Speicherlecks	100
Gemeinsame Nutzung von Speicher	101
Together konfigurieren	102
Refactoring: Parameter ändern	103
Refactoring: Interfaces extrahieren	104
Refactoring: Methoden extrahieren	105
Refactoring: Superklasse extrahieren	106
Refactoring: Inline-Variablen erstellen	107
Refactoring: Felder einführen	108
Refactoring: Variablen einführen	109
Refactoring: Member verlagern	110
Refactoring: Member in die übergeordnete oder abgeleitete Klasse verschieben	111
Refactoring: Sicheres Löschen	112
Together – Anleitungen	113
Die Online-Hilfe verwenden	114
Diagramme mit Hinweisen versehen	115
Diagramme erstellen	116
Notation für Diagramme ändern	117
Raster und andere Optionen für das Erscheinungsbild verwenden	118
Das Profil "UML in Farbe" verwenden	119
Modellelemente ausrichten	120
Den Typ einer Beziehung ändern	121
Diagramme schließen	122
Modellelemente kopieren und einfügen	123
Diagramme löschen	124
Hyperlinks in Diagrammen	125
Automatisches Layout für Diagramme	127
Modellelemente verschieben	128

Diagramme umbenennen	129
Beziehungen umlenken	130
Größe von Modellelementen ändern	131
Modellelemente auswählen	132
Element-Stereotyp zuweisen	133
Drag&Drop	134
Benutzereigenschaften verwenden	135
Diagramme als Grafik exportieren	136
Beziehungslinien mit Umlenkpunkten erstellen	137
Mehrere Elemente erstellen	138
Verknüpfungen erstellen	139
Eine einfache Beziehung erstellen	141
Einzelnes Modellelement erstellen	142
Diagramme drucken	143
Diagrammdateien der Versionskontrolle unterstellen	144
In Diagrammen suchen	146
Quelltext nach Verwendungen durchsuchen	147
Aktivität für einen Zustand erzeugen	148
UML 1.5-Aktivitätsdiagramme erstellen	149
Klassifizierer instantiiieren	150
UML 1.5-Komponentendiagramme erstellen	151
UML 1.5-Verteilungsdiagramme erstellen	152
Konditionale Blöcke hinzufügen	153
Objekten einen Klassifizierer zuordnen	154
Nachrichtenbeziehungen verzweigen	156
Zwischen UML 1.5-Sequenz- und -Kollaborationsdiagrammen umschalten	157
Mit UML 1.5-Nachrichten arbeiten	158
UML 1.5-Zustandsdiagramme erstellen	160

Pins erstellen	161
UML 2.0-Aktivitätsdiagramme erstellen	162
Aktionen in einer Aktivität gruppieren	164
Mit Objektfluss- und Kontrollflussbeziehungen arbeiten	165
UML 2.0-Komponentendiagramme erstellen	167
Delegationskonnektoren erstellen	168
Eine interne Struktur für einen Knoten erzeugen	169
Ports erstellen	170
Ein referenziertes Part erstellen	171
Mit Kollaborationsverwendungen arbeiten	172
UML 2.0-Verteilungsdiagramme erstellen	174
Lebenslinien mit Klassifizierern verknüpfen	175
Ausführungs- und Aufrufspezifikationen kopieren und einfügen	176
Sequenz- oder Kommunikationsdiagramm aus einer Interaktion erstellen	177
Zustandsinvarianten erstellen	178
UML 2.0-Sequenzdiagramme oder -Kommunikationsdiagramme erstellen	179
Verknüpfung zu einer Interaktion in ein Interaktionsdiagramm einfügen	181
Mit UML 2.0-Nachrichten arbeiten	182
Mit kombinierten Fragmenten arbeiten	183
Frames verankern	185
Übergang oder Zustand mit einer Aktivität verknüpfen	186
Überwachungsbedingung für einen Übergang erstellen	187
Historienelement erstellen	188
Member für einen Zustand erzeugen	189
Zustände erstellen	190
UML 2.0-Zustandsmaschinendiagramme erstellen	191
Navigation im Diagramm mit der Übersicht	192
Modellelemente aus- und einblenden	193

Ansichtsfilter verwenden	194
Diagramme zoomen	195
OCL-Einschränkungen erstellen	196
OCL-Ausdrücke bearbeiten	197
OCL-Einschränkungen ein- und ausblenden	198
Mit komplexen Zuständen arbeiten	199
Verzögertes Ereignis erstellen	200
Interne Übergänge erstellen	201
Mehrfachübergänge erstellen	202
Selbstübergänge erstellen	203
Eintritts- und Austrittsaktionen festlegen	204
Mit Instanzspezifikationen arbeiten	205
Mit bereitgestellten und erforderlichen Interfaces arbeiten	207
Assoziationsklasse erstellen	208
Innere Klassifizierer erstellen	209
Klassendiagramme als Ansichten verwenden	210
Mit Interfaces arbeiten	211
Einem Container ein Member hinzufügen	212
Das Erscheinungsbild von Abschnitten ändern	213
Das Erscheinungsbild von Interfaces ändern	214
Mit Konstruktoren arbeiten	215
Mit Feldern arbeiten	216
Mit Beziehungen arbeiten	217
Nachrichtenbeziehungen einer Methode zuordnen	218
Eine Diagrammabfolge erstellen	220
Erweiterungspunkte erstellen	221
Eine Anwendungsfallhierarchie erstellen	222
Funktion zur Dokumentationserzeugung konfigurieren	223

Projektdokumentation erzeugen	224
Mit Namespaces und Paketen arbeiten	225
Beziehungen nach Pattern erstellen	226
Modelelemente nach Pattern erstellen	227
Stub-Implementierungs-Pattern verwenden	228
Hinzufügen von Teilnehmern zu FCC-Pattern	230
Ein Pattern erstellen	231
Entfernen von FCC-Pattern aus dem Modell	232
Pattern exportieren	233
Vorhandene Pattern importieren	234
Gemeinsame Nutzung von Pattern	235
Pattern an Verknüpfungen zuweisen	236
Verknüpfungen, Ordner und Pattern-Hierarchien kopieren und einfügen	237
Ordner erstellen	238
Verknüpfung zu einem Pattern erstellen	239
Virtuelle Pattern-Hierarchie erstellen	240
Verknüpfungen, Ordner und Pattern-Hierarchien löschen	241
Eigenschaften bearbeiten	242
Pattern-Organizer öffnen	243
Änderungen in der Pattern-Registrierung speichern	244
Pattern sortieren	245
Den Pattern-Organizer verwenden	246
Die Pattern-Registrierung verwenden	247
Together-Unterstützung für Projekte aktivieren	248
Ein Projekt erstellen	249
Projekt in das XMI-Format exportieren	250
Projekt im IBM Rational Rose-Format (MDL) importieren	251
In TCC oder TAR erstellte Projekte importieren	252

In TVS, TEC, TJB oder TPT erstellte Projekte importieren	254
Projekt im XMI-Format importieren	256
Vorhandenes Projekt für die Modellierung öffnen	257
Gemeinsame Nutzung eines Projekts in TCC/TAR und RAD Studio	258
Modellansicht, Diagrammansicht und Quelltext synchronisieren	262
Designprojekte in Quelltext umwandeln	264
Fehlerbehebung an Modellen	265
Mit referenzierten Projekten arbeiten	266
Audit-Ergebnisse exportieren	267
Audit-Ergebnisse drucken	268
Audits ausführen	269
Audit-Ergebnisse anzeigen	270
Mit Audit-Sets arbeiten	271
Metrikdiagramme erstellen	272
Metriken ausführen	273
Metrikergebnisse anzeigen	274
Mit Metrik-Sets arbeiten	276
RAD StudioDialog-Hilfe	277
Code-Visualisierungsdiagramm	278
Diagramm als Bild exportieren	279
Komponenten-Template erzeugen	280
Experte "VCL-Komponente importieren"	281
Installieren	283
Komponente installieren	284
Installierte ActiveX-Komponenten	285
Suchpfade für Assemblierung	286
Installierte .NET-Komponenten	287
.NET VCL-Komponenten	288

Packages	289
Neue Komponente	290
Experte für neue VCL-Komponenten	291
Felder hinzufügen	293
Lokale Daten zuweisen	294
Spaltensammlungs-Editor	295
Bedingungssammlungs-Editor	296
Beziehungskollektions-Editor	297
Tabellensammlungs-Editor	298
Anweisungstext-Editor	299
Anweisungstext-Editor	300
Datenadapter konfigurieren	301
Verbindungseditor	302
Verbindungseditor	303
Editor für Verbindungs-Strings (ADO)	304
Datenadapter-Konfiguration DataSet	305
Datenadapter-Konfiguration Daten in der Vorabansicht	306
Datenbank-Editor	307
Datenbankformular-Experte	308
Eigenschaften der Datenmenge	309
Treibereinstellungen	310
Feldverbindungs-Designer	311
Felder-Editor	312
Fremdschlüsselbedingung	313
Datenmenge erzeugen	314
Datenbankkomponenten-Editor (Dialogfeld)	315
Transaktions-Editor (Dialogfeld)	316
IBUpdateSQL- und IBDataSet-Editor (Dialogfeld)	317

Neue Verbindung	319
Neues Feld	320
Beziehung	321
Verbindung umbenennen	322
SortFields-Editor	323
SQL-Monitor	324
Stored Procedure (Dialogfeld)	325
TableMappings Collection-Editor	326
Eindeutige Einschränkung	327
Datenbankevolution	328
EcoSpace-Designer	329
Schema erzeugen	330
OCL und ECO-AktionsspracheAusdruckseditor	331
Packages für den EcoSpace auswählen	332
Referenzierte ECO-Packages	333
Experte zum Kapseln von Datenbanken	334
Ausrichtung	336
Erstellungsfolge	337
Tabulator-Reihenfolge bearbeiten	338
Skalierung	339
Größe	340
Fehleradresse nicht gefunden	341
Eine andere Datei mit dem Namen <dateiname> befindet sich bereits im Suchpfad	342
Aufgrund des Hard-Mode konnte nicht gestoppt werden	343
Fehler beim Erstellen des Berichts: <prozess> (<fehlercode>)	344
Eine Komponentenklasse mit dem Namen <name> existiert bereits	345
Ein Feld oder eine Methode mit dem Namen <name> existiert bereits	346
Das Projekt enthält bereits ein Formular bzw. ein Modul mit dem Namen <name>	347

Falsche Feld-Deklaration in der Klasse <klassenname>	348
Das Feld <feld> hat keine korrespondierende Komponente.Deklaration entfernen?	349
Das Feld <feld> sollte vom Typ <typ1> sein, ist jedoch als <typ2> deklariert. Deklaration korrigieren?	350
Deklaration der Klasse <klassenname> fehlt oder ist falsch	351
Modulkopf fehlt oder ist fehlerhaft	352
IMPLEMENTATION-Abschnitt fehlt oder ist falsch	353
Ungenügend Speicher zum Starten	354
Haltepunkt wurde in eine Zeile gesetzt, die weder Quelltext noch Debug-Informationen enthält.	355
<IDname> ist kein gültiger Bezeichner	356
Die Bibliothek <bibliothek> ist schon geladen, wahrscheinlich aufgrund eines ungültigen Programmabbruchs. Ihr System könnte instabil sein, deshalb sollten Sie Windows jetzt beenden und neu starten.	357
Falsche Methoden-Deklaration in der Klasse <klassenname>	358
Eine Implementation der Methode <methodenname> kann nicht gefunden werden	359
Die <methodenname>, aufgerufen von <formularname>.<ereignisname> hat eine inkompatible Parameterliste. Soll der Aufruf entfernt werden?	360
Die Methode <methodenname>, aufgerufen von <formularname>, existiert nicht. Soll der Aufruf entfernt werden?	361
Es wurde keine Anweisung für die aktuelle Zeile generiert	362
Eigenschaft und Methode <methodenname> sind nicht kompatibel	363
<dateiname.PAS> oder <dateiname.DCU> kann im aktuellen Suchpfad nicht gefunden werden	364
Source has been modified. Neu compilieren und binden?	365
Symbol <symbol> wurde nicht gefunden.	366
Der Debugger läuft gerade. Beenden?	367
Die Uses-Klausel fehlt oder ist falsch.	368
Ungültiges Ereignisprofil <name>	369
Active Form-Experte	370
Active-Server-Objekt-Experte	371
Hinzufügen	373
Experte für Automatisierungsobjekte	374

COM-Objekt-Experte	376
Dialogfenster für COM+ Ereignisschnittstelle-Auswahl	377
Experte für COM+ Ereignisobjekte	378
COM+-Subskriptionsobjekt-Experte	379
Neues Menü anpassen	380
Namen der Zielfile ändern	381
FTP-Verbindungsoptionen	382
Deploymentmanager	383
Von RAD Studio erzeugte Dateien	384
Experte für Interface-Auswahl	386
Neue ASP.NET-Anwendung	387
Neue Konsolenanwendung	388
Experte für neues DBWeb Control	389
Neue Dynamische Link-Bibliothek	390
Objektgalerie	391
Neue Anwendung	392
Neues Remote-Datenmodulobjekt	393
Neues Thread-Objekt	394
Öffnen	395
Package	396
Druckauswahl	397
Projekt-Upgrade	398
Projekte aktualisieren	399
Experte für externes Datenmodul	400
Satellite-Assemblierungsexperte	402
Speichern unter	403
Verzeichnis auswählen	404
Neuen Web-Service hinzufügen	405

SOAP-Datenmodul-Experte	406
Neue SOAP-Server-Anwendung	407
Transaktionaler Objektexperte	408
Unit verwenden	410
Neue Web-Server-Anwendung	411
Neuen Web-Service hinzufügen	412
Seitenoptionen des Anwendungsmoduls/Neues WebSnap-Seitenmodul	413
Neue WebSnap-Anwendung	414
Neues WebSnap-Datenmodul	416
Web-Anwendungskomponenten	417
WSDL: Optionen für den Import	418
Experte fÃ¼r den WSDL-Import	420
Optionen des Experten für Datenbindungen	422
Experte für XML-Datenbindung, Seite 1	423
Experte für XML-Datenbindung, Seite 2	424
Experte für XML-Datenbindung, Seite 3	425
A (Anker) HTML-Element	426
Unit	428
DIV HTML-Element	429
HR HTML-Element	430
IMG HTML-Element	432
INPUT HTML-Element	434
SELECT HTML-Element	437
SPAN HTML-Element	439
TABLE HTML-Element	440
TEXTAREA HTML-Element	442
User Control einfügen	444
Bild einfügen	445

Eingabe einfügen	446
Tabelle einfügen	447
Farbe auswählen	448
EJBs aus Liste auswählen	449
Testfall-Experte	450
Testfall-Experte	451
Testprojekt-Experte	452
Testprojekt-Experte	453
Erweiterte Datenbindung	454
AutoFormat	455
Auflistungs-Editor	456
Datenbindungen	457
Dynamische Eigenschaften	458
Eigenschaften	459
<AliasDef>	460
<AttributeDef>	461
<ClassDef>	462
<Classes>	463
<ConstantColumn>	464
<DiscriminatorColumn>	465
<DiscriminatorDef>	466
<DiscriminatorImpl>	467
<DiscriminatorValue>	468
<Globals>	469
<KeyColumn>	470
<KeyDef>	471
<KeyImpl>	472
<SingleLinkDef>	473

COM-Importe	474
.NET-Assemblierungen	475
Projektreferenzen	476
Der Objektablage hinzufügen	477
UDDI-Browser	478
ASP-Deployment-Manager	479
Package ändern	481
Sprachen hinzufügen	482
Sprachen entfernen	483
Aktive Sprache festlegen	484
Neuer Kategorienname	485
Entwurfs-Packages hinzufügen	486
Laufzeit-Package hinzufügen	487
Suchpfad für Symboltabelle hinzufügen	488
Anwendung	489
Anwendung	490
Anwendung (Visual Basic)	491
Optionsgruppe anwenden	492
Cassini konfigurieren	493
Virtuelles Verzeichnis konfigurieren	494
ASP .NET	495
Manager für Build-Konfigurationen	496
Build-Ereignisse	497
CodeGuard	498
Compiler	499
Compiler	501
Compiler-Meldungen	507
Compiler (Visual Basic)	508

Komponente	509
Debugger	510
Beschreibung	511
Verzeichnisse/Bedingungen	512
Debugger-Umgebung	513
Formulare	514
Linker	515
Projektoptionen	517
Packages	519
Pre-Build- oder Post-Build-Ereignis	520
Signatur	521
Debugger-Symboltabellen	522
Versionsinformationen	523
Ordner- oder Verzeichnisansicht hinzufügen	524
ATL	525
C++-Compiler Erweiterte Compilierung	526
C++-Compiler C++-Kompatibilität	529
C++Compiler C++-Compilierung	531
C++ Compiler Debugging	534
C++-Compiler C++-Kompatibilität	536
C++-Compiler Allgemeine Compilierung	538
C++-Compiler	543
C++-Compiler Optimierungen	544
C++ Compiler Ausgabe	545
C++-Compiler Pfade und Definitionen	546
C++-Compiler Vorcompilierte Header	547
C++-Compiler Warnungen	549
Ressourcen-Compiler	550

Ressourcen-Compiler Optionen	551
Ressourcen-Compiler Pfade und Definitionen	552
Delphi-Compiler Compilierung	553
Delphi-Compiler	559
Delphi-Compiler Weitere Optionen	560
Delphi-Compiler Pfade und Definitionen	562
Delphi-Compiler Warnungen	563
Option suchen	564
Linker Linken	565
Linker	568
Linker Ausgabeoptionen	570
Linker Warnungen	575
Bibliotheksverwaltung	576
Listeneditor	577
Pfade und Definitionen	578
Projekteigenschaften	580
Turbo Assembler	581
Turbo Assembler Optionen	582
Turbo Assembler Pfade und Definitionen	585
Turbo Assembler Warnungen	586
Nicht verfügbare Optionen	587
Information	596
Aus dem Projekt entfernen	597
Optionen	598
Symbol wählen	600
Optionen für Distribution über das Web	601
Durchsuchen (Dialogfeld)	603
Anderes Symbol	604

Farbeditor	605
DDE-Info (Dialogfeld)	607
Vorlagen löschen	608
Filter-Editor	609
Schriftarten-Editor	610
Aktionsmanager-Editor	611
Aktionslisten-Editor	613
Seite hinzufügen (Dialogfeld)	615
Auflistungs-Editor	616
Seite bearbeiten (Dialogfeld)	618
IconView-Eintragseditor	619
Bilderlisten-Editor	621
ListView Items Editor	623
Standardaktionsklassen (Dialogfeld)	624
String-Listen-Editor	625
TreeView-Eintragseditor	626
Wertlisten-Editor	627
Eingabemasken-Editor	628
Template einfügen	630
Objekt einfügen	631
Laden eines Bildes zur Entwurfszeit	633
Bild laden (Dialogfeld)	634
String-Liste laden	635
Maskentext-Editor	636
Arbeitsblatt-Editor	637
Öffnen (Dialogfeld)	638
Inhalte einfügen (Dialogfeld)	639
Bild-Editor	640

Bild speichern unter	641
String-Liste speichern (Dialogfeld)	642
Menü auswählen	643
Parameter ändern	644
Parameter hinzufügen	645
Namespace hinzufügen	646
Felder deklarieren	647
Variablen deklarieren	648
Methode extrahieren	649
Ressourcen-String extrahieren	650
Unit suchen	651
Refactorings	652
Symbol umbenennen (C++)	653
Umbenennen (C#)	654
Umbenennen (Delphi)	655
Mit Prozess verbinden	656
Änderung	657
Benachrichtigung über Debugger-Exception	658
Debug-Inspektor	659
Auswerten/Ändern	661
Package-Import suchen	663
Untersuchen	664
Prozess laden Umgebungsblock	665
Prozess laden Lokal	666
Prozess laden Extern	667
Prozess laden Symboltabellen	668
Neuer Ausdruck	669
Der Debugger läuft gerade. Beenden?	670

Typumwandlung	671
Eigenschaften Darstellung überwachter Ausdrücke	672
Suchen	674
In Dateien suchen	675
Überblick zum Suchen von Referenzen	676
Adresse eingeben	677
Zu Zeilennummer gehen	678
Suchen und Ersetzen	679
Together- Optionen für Sequenzdiagramm-Roundtrips	680
Systemmakros	681
Neues Diagramm hinzufügen (Dialogfeld)	682
Benutzereigenschaften hinzufügen/entfernen (Dialogfeld)	683
Parameter ändern (Dialogfeld)	684
Zielprojekt auswählen/Quellprojekt auswählen (Dialogfeld)	685
Bearbeiten der Hyperlinks für Diagramm (Dialogfeld)	686
Diagramm als Bild exportieren (Dialogfeld)	687
Interface extrahieren/Superklasse extrahieren (Dialogfelder)	688
Methode extrahieren (Dialogfeld)	689
Dokumentation erzeugen (Dialogfeld)	690
Inline für Variable (Dialogfeld)	691
Feld einführen (Dialogfeld)	692
Variable einführen (Dialogfeld)	693
Modellunterstützung	694
Verlagern (Dialogfeld)	695
Together-Optionen (Dialogfeld)	696
Audit drucken (Dialogfeld)	698
Diagramm drucken (Dialogfeld)	699
Member in übergeordnete Klasse verschieben/Member in abgeleitete Klasse verschieben (Dialogfelder)	700

QS-Audits (Dialogfeld)	701
QS-Metriken (Dialogfeld)	703
Sicheres Löschen (Dialogfeld)	705
Audit-Ergebnis speichern (Dialogfeld)	706
Verwendung suchen (Dialogfeld)	707
Dialogfelder zur Elementauswahl	708
Auswahlmanager	709
XMI-Export (Dialogfeld)	710
XMI-Import (Dialogfeld)	711
CodeGuard-Konfiguration	712
Tools-Optionen	715
Objekt-Info bearbeiten	716
Bearbeitungs-Tool	717
Visual Studio-Projekt exportieren	718
Versionsverwaltung	719
Objektablage	721
Bereich für Exception hinzufügen	722
Sprach-Exception hinzufügen	723
Aktualisierung durchführen	724
ASP.NET	725
CodeGear-Debugger	726
Code Insight	728
Farben	730
Quelltextvorlagen	731
Farbe	732
Pfade und Verzeichnisse (C++)	733
Typbibliothek (C++)	734
Debugger-Optionen	735

VCL-Designer	736
Bibliothek	737
Anzeigeoptionen	738
ECO - Allgemeine Optionen	739
Editor-Optionen	741
Umgebungsoptionen	743
Umgebungsvariablen	744
Ereignisprotokoll-Optionen	745
Explorer	747
Generische sortierte Liste	748
HTML/ASP.NET-Optionen	749
HTML-Formatierung	750
HTML Tidy-Optionen	751
Tastaturbelegung	752
Sprach-Exceptions	753
Native BS-Exceptions	754
Neue Tags	755
Systemvariable überschreiben/Neue Anwendervariable/Anwendervariable bearbeiten	756
Objektinspektor	757
Quelloptionen	758
Farben	759
Tool-Palette	760
Farbe	761
Schrift	762
Formular-Designer	763
Packages	764
Optionen für Übersetzungs-Tools	765
Wörterbuch	766

Übersetzungswörterbuch	767
Typbibliothek (Delphi)	768
WebSnap	769
Windows Forms-Designer	770
Tools-Eigenschaften	771
XML-Mapper	772
Der Objektablage hinzufügen	775
Code-Explorer	776
Symbolleisten anpassen	777
Daten-Explorer	778
Modul hinzufügen	781
Gruppe überwachter Ausdrücke hinzufügen	782
Liste der Haltepunkte	783
Aufruf-Stack	785
CPU-Fenster	786
Neuen Wert eingeben	790
Such-Bytes eingeben	791
Fenster Ereignisprotokoll	792
Kommentar zu Ereignisprotokoll hinzufügen	793
FPU	794
Lokale Variablen	797
Fenster Module	798
Datei nicht gefunden	800
Fenster Thread-Status	801
Fenster Liste überwachter Ausdrücke	802
Gespeicherten Desktop löschen	804
Symbolleiste Desktop	805
Datei-Browser	806

Dem Wörterbuch hinzufügen	807
Fenster: Meldungen	808
Objektinspektor	809
Projektverwaltung	810
Desktop speichern	814
Debug-Desktop einstellen	815
Strukturansicht	816
To-Do-Liste	817
To-Do-Eintrag hinzufügen oder bearbeiten	818
To-Do-Liste filtern	819
Tabelleneigenschaften	820
Tool-Palette	821
Translation-Manager	823
Mehrzeilen-Editor	825
Typbibliothekseditor	826
Formular anzeigen	830
Units anzeigen	831
Fensterliste	832
Assemblierungs-Metadaten-Explorer	833
Typbibliothek-Explorer	835
Suchen	837
Quelltexthaltelpunkt hinzufügen	839
Reference	841
Delphi-Referenz	842
Delphi-Sprachreferenz	842
Übersicht über Delphi	842
Programme und Units	847
Grundlegende syntaktische Elemente	857

Datentypen, Variablen und Konstanten	887
Prozeduren und Funktionen	930
Klassen und Objekte	948
Standardroutinen und E/A	988
Bibliotheken und Packages	997
Objekt-Interfaces	1006
Speicherverwaltung auf der Win32-Plattform	1017
Ablaufsteuerung	1030
Verwendung des integrierten Assemblers (nur Win32)	1034
Benutzerdefinierte Attribute in .NET	1049
Liste der Compiler-Direktiven	1052
Felder ausrichten (Delphi)	1052
Anwendungstyp (Delphi)	1053
Assert-Direktiven (Delphi)	1053
AUTOBOX-Direktive (Delphi für .NET)	1053
Boolesche Kurzauswertung (Delphi-Compiler-Direktive)	1054
Debug-Informationen (Delphi)	1055
DEFINE-Direktive (Delphi)	1055
DENYPACKAGEUNIT-Direktive (Delphi)	1056
Beschreibung (Delphi)	1056
DESIGNONLY-Direktive (Delphi)	1056
ELSE (Delphi)	1057
ELSEIF (Delphi)	1057
ENDIF-Direktive	1057
Erweiterung für ausführbare Dateien (Delphi)	1058
Symbole exportieren (Delphi)	1058
Erweiterte Syntax (Delphi)	1058
Externe Symbole (Delphi)	1059
Prüfung von Fließkomma-Exceptions (Delphi)	1059
Hinweise (Delphi)	1060
HPP-Ausgabe (Delphi)	1060
IFDEF-Direktive (Delphi)	1060
IF-Direktive (Delphi)	1061
IFEND-Direktive (Delphi)	1062
IFNDEF-Direktive (Delphi)	1062
IFOPT-Direktive (Delphi)	1063
Image-Basisadresse	1063
Implizites Erstellen (Delphi)	1064
Importierte Daten	1064

Datei einbinden (Delphi)	1064
E/A-Prüfung (Delphi)	1065
Direktiven für Bibliotheken und gemeinsame Objekte (Delphi)	1065
Objektdatei linken (Delphi)	1066
Lokale Symbolinformationen (Delphi)	1066
MESSAGE-Direktive (Delphi)	1067
\$METHODINFO-Direktive (Delphi)	1067
Mindestgröße für Enum-Typen (Delphi)	1068
Offene String-Parameter (Delphi)	1068
Optimierung (Delphi)	1068
Überlaufprüfung (Delphi)	1069
Pentium-sichere FDIV-Operationen (Delphi)	1069
NODEFINE	1070
NOINCLUDE (Delphi)	1070
Bereichsüberprüfung	1070
Real48-Kompatibilität (Delphi)	1071
Abschnitte (Delphi und C#)	1071
Ressourcen-Datei (Delphi)	1072
RUNONLY-Direktive (Delphi)	1072
Laufzeit-Typinformationen (Delphi)	1073
Typprüfung bei Zeigern (Delphi)	1073
UNDEF-Direktive (Delphi)	1074
Unsicherer Code (Delphi für .NET)	1074
Prüfung von Var-String (Delphi)	1074
Warnmeldungen (Delphi)	1075
Warnungen (Delphi)	1076
Schwaches Packen	1076
Stack-Frames (Delphi)	1077
Schreibbare typisierte Konstanten (Delphi)	1077
PE (Portable Executable) Header-Flags (Delphi)	1078
Reservierter Adressraum für Ressourcen (Delphi)	1078
Delphi Compiler-Fehler	1079
Laufzeitfehler	1079
Interne Fehler beheben	1080
Schwerwiegende Fehler	1081
E/A-Fehler	1083
Betriebssystemfehler	1083
Fehlermeldungen	1084

Together-Glossar	1266
GUI-Komponenten für die Modellierung	1267
Menüs	1267
Tool-Palette	1267
Objektinspektor	1268
Diagrammansicht	1269
Modellansicht	1270
GUI-Komponenten für Pattern	1271
GUI-Komponenten für die Qualitätssicherung	1273
Together-Experten	1275
Neue Projektexperten in Together	1275
Pattern-Experte	1277
Experte zum Erstellen von Pattern	1278
Together-Tastaturkürzel	1278
Together-Konfigurationsoptionen	1280
Konfigurationsebenen	1280
Optionswert-Editoren	1280
Together-Optionskategorien	1281
UML 1.5-Referenz	1294
UML 1.5-Klassendiagramme	1294
UML 1.5-Anwendungsfalldiagramm	1300
UML 1.5-Interaktionsdiagramm	1302
UML 1.5-Zustandsdiagramm	1305
UML 1.5-Aktivitätsdiagramm	1308
UML 1.5-Komponentendiagramm	1310
UML 1.5-Verteilungsdiagramm	1311
UML 2.0-Referenz	1313
UML 2.0-Klassendiagramme	1313
UML 2.0-Anwendungsfalldiagramm	1314
UML 2.0-Interaktionsdiagramm	1315
UML 2.0-Zustandsmaschinendiagramm	1321
UML 2.0-Aktivitätsdiagramm	1323
UML 2.0-Komponentendiagramm	1326
UML 2.0-Verteilungsdiagramm	1328
UML 2.0-Kompositionsstrukturdiagramm	1329
Together – Refactoring-Operationen	1331
Projekttypen und -formate für das Modellieren	1332
Befehlszeilenoptionen	1333
IDE-Befehlszeilenoptionen	1333

Tastenzuordnungen	1335
Standard-Tastaturvorlage	1335
Tastaturvorlage IDE - Klassisch	1338
Tastaturvorlage BRIEF-Emulation	1340
Tastaturvorlage Epsilon-Emulation	1342
Tastaturvorlage Visual Studio-Emulation	1344
Tastaturvorlage Visual Basic-Emulation	1345
Quelltexthaltepunkt hinzufügen/Adresshaltepunkt hinzufügen/Datenhaltepunkt hinzufügen	1349
Compilieren, Build erstellen und Anwendungen ausführen	1351
Einführung	1353
Was ist RAD Studio?	1354
Neuerungen in RAD Studio (Delphi)	1355
Neuerungen in RAD Studio (C++Builder)	1358
Ein erster Blick auf die IDE	1363
Ein Projekt starten	1369
Quelltext-Editor	1373
Hilfe zur Hilfe	1378
Erste Schritte mit Together	1380
Allgemeines zu Together	1380
Beispielprojekt "Cash Sales"	1381
Öffnen des C#-Beispielprojekts "Cash Sales"	1381
UML 1.5-Modellierung mit "Cash Sales"	1381
Pattern in "Cash Sales" verwenden	1382
Audits für "Cash Sales" ausführen	1383
Dokumentation für "Cash Sales" erzeugen	1383
"Cash Sales" ausführen	1384
UML 2.0-Beispielprojekt	1384
Das UML 2.0-Beispielprojekt öffnen	1384
UML 2.0-Beispielprojekt, Structure-Paket	1385
UML 2.0-Beispielprojekt, Behavior-Paket	1386

Anleitungen	1389
Einführende Anleitungen	1390
Compilieren und Builds erstellen – Anleitungen	1391
Debugging – Anleitungen	1392
Quelltext bearbeiten – Anleitungen	1393
Lokalisierung von Anwendungen – Anleitungen	1394
Together-Diagramme – Anleitungen	1395
Together-Dokumentationserzeugung – Anleitungen	1396
Together Object Constraint Language (OCL) – Anleitungen	1397
Together-Pattern – Anleitungen	1398
Together-Projekte – Anleitungen	1399
Together-Qualitätssicherung – Anleitungen	1400
Together-Refactoring – Anleitungen	1401
Testen von Units – Anleitungen	1402
Templates (Fenster)	1403
Entwicklungszyklus steuern	1405
Überblick zum Steuern des Entwicklungszyklus	1406
Mit der SourceControl arbeiten	1408
Benutzeroberflächen entwerfen	1410
Refactoring von Anwendungen	1412
Überblick zum Refactoring	1412
Überblick zum Umbenennen von Symbolen (Delphi, C#, C++)	1413
Überblick zum Extrahieren von Methoden (Delphi)	1414
Refactoring von Quelltext	1415
Überblick zum Extrahieren von Ressourcen-Strings (Delphi)	1417
Refactoring-Operationen in der Vorschau anzeigen und durchführen	1417
Überblick zum Deklarieren von Variablen und Feldern (Delphi)	1419
Überblick zum Suchen von Referenzen (Delphi, C#, C++)	1421
Überblick zum Ändern von Parametern (Delphi)	1422

Sync-Bearbeitungsmodus (Delphi, C#, C++)	1422
Refactoring rückgängig machen (Delphi, C#)	1422
Referenzen suchen	1423
Units suchen und Namespaces verwenden (Delphi, C#)	1424
Anwendungen testen	1425
Überblick über das Testen von Units	1425
Entwickeln von Tests	1427
Überblick über DUnit	1429
Überblick über NUnit	1432
Anwendungen lokalisieren	1437
Anwendungen debuggen	1439
Überblick zum Debuggen	1439
Überblick zum externen Debugger	1441
Deployment von Anwendungen	1443
Anwendungen mit Together modellieren	1446
Überblick zur Modellierung	1446
Together-Projekte – Überblick	1446
Überblick zu Namespaces und Paketen	1446
Überblick über Together-Diagramme	1447
Unterstützte UML-Spezifikationen	1447
Übersicht zu Modellelementen	1448
Überblick zu Hinweisen in Diagrammen	1448
Überblick zu Verknüpfungen	1449
Diagrammformat – Überblick	1449
Überblick zum Diagramm-Layout	1450
Überblick zu Hyperlinks	1450
Überblick über LiveSource	1451
Überblick zur Umwandlung in Quelltext	1452
Überblick zur OCL-Unterstützung	1453
Überblick über Pattern	1454
Überblick zum Refactoring	1456
Überblick über Qualitätssicherungsfunktionen	1456
Überblick zur Dokumentationserzeugung	1458
Überblick zum Importieren und Exportieren	1459
Überblick zur Interoperabilität	1460

Überblick über Build-Konfigurationen (Delphi)	1463
Überblick über Build-Konfigurationen (C++)	1465
Überblick über benannte Optionsgruppen	1469
Targets-Dateien	1471
Index	a

1 Alle Anleitungen

Dieser Abschnitt führt alle Anleitungen für RAD Studio auf. Eine Liste der Anleitungen nach Entwicklungsbereichen finden Sie im Inhaltsverzeichnis unter *Anleitungen*.

1.1 Anwenden der aktiven Build-Konfiguration auf ein Projekt

So übernehmen Sie mit dem Konfigurations-Manager eine aktive Build-Konfiguration für ein Projekt oder mehrere Projekte:

1. Öffnen Sie den Konfigurations-Manager, indem Sie entweder **Projekt**▶**Konfigurations-Manager** wählen oder in der **Projektverwaltung** eine Projektgruppe mit der rechten Maustaste anklicken und im Kontextmenü **Konfigurations-Manager** auswählen.
2. Wählen Sie im Dialogfeld Manager für Build-Konfigurationen aus der Liste Name der Konfiguration eine Konfiguration aus. Die Liste enthält alle verfügbaren Konfigurationen, die Sie in der Projektgruppe erstellt haben, einschließlich der Standardkonfigurationen (Debug und Ausgabe).
3. Wählen Sie aus dem Bereich Verfügbare Projekte das Projekt oder die Projekte aus, für die die ausgewählte Konfiguration verwendet werden soll. Wenn Sie einen Konfigurationsname auswählen, werden in der Liste Verfügbare Projekte die Namen der Projekte angezeigt, die dieser Konfiguration zugeordnet sind. Die Liste Verfügbare Projekte zeigt auch für jedes Projekt die Aktive Konfiguration.
4. Um die in Name der Konfiguration ausgewählte Konfiguration als aktive Konfiguration für die ausgewählten Projekte zuzuweisen, klicken Sie auf Übernehmen.

Tip: Um alle verfügbaren Projekt auszuwählen, klicken Sie auf Alles markieren.

Tip: Um die Auswahl in der Liste Verfügbare Projekte aufzuheben, klicken Sie auf Alles löschen.

So aktivieren Sie in der Projektverwaltung eine Build-Konfiguration:

1. Doppelklicken Sie in der **Projektverwaltung** entweder auf den Namen einer Build-Konfiguration (wie z.B. Debug oder Ausgabe), oder klicken Sie den Namen mit der rechten Maustaste an, und wählen Sie aus dem Kontextmenü **Aktivieren**.
2. Der Name der Konfiguration wird in Fettschrift dargestellt, um anzuzeigen, dass es sich um die aktuell aktive Build-Konfiguration handelt.

Siehe auch

Überblick zu MSBuild (siehe Seite 1461)

Überblick zu Build-Konfigurationen (siehe Seite 1463)

Compiler (siehe Seite 1351)

Erstellen von benannten Build-Konfigurationen für C++ (siehe Seite 6)

Manager für Build-Konfigurationen (siehe Seite 496)

1.2 Packages erstellen

Sie können in RAD Studio auf einfache Weise Packages erstellen und in Projekte aufnehmen.

So erstellen Sie ein neues Package:

1. Wählen Sie **Datei > Neu > Weitere**, um die Objektgalerie zu öffnen.
2. Wählen Sie je nach Projekttyp den Knoten Delphi-Projekte, Delphi für .NET-Projekte oder C++Builder-Projekte.
3. Doppelklicken Sie auf das Symbol Package. Daraufhin werden ein neues, leeres Package und der zugehörige Eintrag in der Projektverwaltung sowie die folgenden beiden Ordner erstellt: ein Ordner mit der Bezeichnung Contains und ein Ordner mit der Bezeichnung Requires.

Anmerkung: Wenn Sie für das Package erforderliche Dateien hinzufügen möchten, müssen Sie compilierte Packages (**.dcpil**, **.dll**) in den Ordner Required kopieren. Nicht compilierte Quelltextdateien (**.pas**, **.cpp**, **.h**) fügen Sie dem Ordner Contains hinzu.

4. Markieren Sie den Package-Namen in der Projektverwaltung.
5. Klicken Sie mit der rechten Maustaste, um das Kontextmenü zu öffnen, und wählen Sie Hinzufügen, um das Dialogfeld Hinzufügen anzuzeigen.
6. Klicken Sie auf die Schaltfläche Durchsuchen, und suchen Sie nach den gewünschten Dateien.
7. Markieren Sie eine oder mehrere Dateien, und klicken Sie auf Öffnen.
8. Klicken Sie auf OK. Damit werden die ausgewählten Dateien dem Package hinzugefügt.
9. Wählen Sie **Projekt > <Package-Name> erzeugen**, um das Package zu compilieren.

So fügen Sie ein Package in ein Projekt ein:

1. Wählen Sie **Datei > Neu > Weitere > VCL-Formularanwendung**.
2. Markieren Sie den Package-Namen in der Projektverwaltung.
3. Klicken Sie mit der rechten Maustaste, um das Kontextmenü anzuzeigen.
4. Wählen Sie Hinzufügen.
5. Klicken Sie auf die Schaltfläche Durchsuchen, um die Package-Datei ausfindig zu machen.
6. Markieren Sie die Datei, und klicken Sie auf Öffnen.
7. Klicken Sie auf OK. Damit wird das Package dem Projekt hinzugefügt.
8. Wählen Sie **Projekt > <Projektname> erzeugen**, um das Projekt zu compilieren.

So fügen Sie der Tool-Palette ein Komponenten-Package hinzu:

1. Wählen Sie **Komponenten > Installierte .NET-Komponenten**.
2. Aktivieren Sie die Registerkarte .NET VCL-Komponenten.
3. Klicken Sie auf Hinzufügen.
4. Suchen Sie die Package-Datei, die Sie der Tool-Palette hinzufügen möchten.
5. Klicken Sie auf Öffnen. Daraufhin werden die im Package verfügbaren Komponenten angezeigt.
6. Klicken Sie auf OK. Die Komponenten werden in der Tool-Palette angezeigt.

Siehe auch

Compilieren (siehe Seite 1351)

1.3 Compilieren von Entwurfszeit-Packages, die Delphi-Quelltextdateien enthalten

C++Builder unterstützt das Compilieren von Entwurfszeit-Packages, die Delphi-Quelltextdateien enthalten. Falls jedoch eine Delphi-Quelltextdatei einen Verweis auf von der IDE bereitgestellte Entwurfszeit-Units, wie z.B. DesignIntf, DesignEditors oder ToolsAPI aus der Datei DesignIDE100.bpl enthält, müssen Sie Vorkehrungen treffen, damit die Verweise vom C++Builder-Package aufgelöst werden können.

So compilieren Sie Entwurfszeit-Packages, die Delphi-Quelltextdateien enthalten:

1. Der Delphi-Compiler muss Units in dem Package DesignIDE auflösen können. Um dies zu aktivieren, wählen Sie im C++Builder-Package-Projekt **Projekt**▸**Optionen**▸**Delphi-Compiler**▸**Weitere Optionen**.
2. Fügen Sie dem Feld Weitere Optionen `-LUDesignIDE` hinzu.
3. Der C++-Linker muss für das Linken den Verweis in der compilierten Entwurfszeit-.obj auflösen können. Um dies zu ermöglichen, wählen Sie im C++Builder-Package-Projekt die Registerkarte **Projekt**▸**Dem Projekt hinzufügen**▸**Erfordert**.
4. Geben Sie in das Feld Name `designide.bpi` ein, und klicken Sie auf OK.

Sie können nun Ihr C++Builder-Package compilieren und installieren.

Siehe auch

[Entwurfszeit-Packages](#)

1.4 Erstellen von Build-Ereignissen

Auf der Registerkarte Build-Ereignisse können Sie eine Liste mit Ereignissen erstellen, die zu verschiedenen Zeitpunkten während des Build-Prozesses auftreten. Abhängig vom Projekttyp lassen sich Pre-Build-, Pre-Link- und Post-Build-Ereignisse festlegen. Ereignisse werden für alle Zeitpunkte auf genau dieselbe Weise hinzugefügt.

So erstellen Sie eine Liste mit Build-Ereignissen:

1. Wählen Sie **Projekt**▸**Optionen**▸**Build-Ereignisse**.
2. Klicken Sie in die Liste Befehle der Build-Phase (Pre-Build, Pre-Link oder Post-Build), zu der Sie Ereignisse hinzufügen möchten.
3. Geben Sie die Build-Befehle ein (jeweils einen Befehl pro Zeile), und drücken Sie nach jedem Befehl die Taste Eingabe. Befehle bestehen aus beliebigen, gültigen DOS-Befehlen, wie z.B.: `copy $(<AusgabePfad>)`
`c:\Built\$(<AusgabeName>)`
4. Zum Anzeigen eines Dialogfeldes, das Sie bei der Eingabe der Befehle unterstützt, klicken Sie auf Bearbeiten. Geben Sie in das Feld Anweisungen dieses Dialogfeldes die Build-Befehle ein. In diesem Feld können Sie die Befehle bearbeiten.
5. Zum Hinzufügen eines vordefinierten Makros blättern Sie in der Liste Makros zu dem gewünschten Makro und doppelklicken auf den Makronamen.
6. Wenn Sie das Hinzufügen von Befehlen beendet haben, klicken Sie auf OK.
7. Wenn Sie auch für die anderen Build-Phasen Befehle hinzugefügt haben, klicken Sie wiederum auf OK.
8. Die Befehle und deren Ergebnisse werden im Bereich Ausgabe des Fensters Meldungen angezeigt.

Siehe auch

Build-Ereignisse (Dialogfeld) (siehe Seite 497)

Pre-Build- oder Post-Build-Ereignis (Dialogfeld) (siehe Seite 520)

1.5 Erstellen von benannten Build-Konfigurationen für C++

So erstellen Sie eine neue Build-Konfiguration:

1. Suchen Sie in der Projektverwaltung den Knoten **Build-Konfigurationen**. Dieser Knoten repräsentiert die Einstellungen der Build-Konfiguration **Base**. Der Knoten enthält alle Build-Konfigurationen.
2. Zum Erstellen einer neuen Build-Konfiguration auf der Basis einer vorhandenen Build-Konfiguration klicken Sie den Namen der Konfiguration (wie z.B. **Debug** oder **Ausgabe**) mit der rechten Maustaste an und wählen **Neue hinzufügen**.
3. Die neue Konfiguration erscheint in der Projektverwaltung; sie erhält den Namen Konfiguration1, wenn es sich um die erste neue Konfiguration handelt. Neue Konfigurationen werden fortlaufend nummeriert.
4. Zum Umbenennen der neuen Konfiguration klicken Sie den Namen mit der rechten Maustaste an und wählen Umbenennen.

So ändern Sie eine vorhandene Build-Konfiguration:

1. Wählen Sie **Projekt>Optionen** und suchen Sie das Feld Build-Konfiguration, das erste Feld auf allen Build-bezogenen Seiten, wie z.B. der Seite Pfade und Definitionen.
2. Wählen Sie auf einer beliebigen Seite, die das Feld Build-Konfiguration enthält, den Name der gewünschten Build-Konfiguration aus.
3. Passen Sie die Projektoptionen auf allen Build-bezogenen Seiten entsprechend Ihren Bedürfnissen an.
4. Klicken Sie zum Speichern auf **OK**.

Tip: Um eine Build-Konfiguration für ein Projekt oder eine Projektgruppe zu übernehmen, klicken Sie in der Projektverwaltung eine Konfiguration mit der rechten Maustaste an und wählen Aktivieren, oder wählen Sie **Projekt>Konfigurations-Manager** und klicken auf **Übernehmen**.

Siehe auch

Überblick zu MSBuild (siehe Seite 1461)

Überblick über Build-Konfigurationen (C++) (siehe Seite 1465)

Manager für Build-Konfigurationen (siehe Seite 496)

Übernehmen der aktiven Build-Konfiguration (siehe Seite 2)

Arbeiten mit benannten Optionsgruppen (C++) (siehe Seite 15)

Festlegen von C++-Standardprojektoptionen (siehe Seite 76)

1.6 Erstellen von benannten Build-Konfigurationen für Delphi

So erstellen Sie eine neue Build-Konfiguration:

1. Suchen Sie in der Projektverwaltung den Knoten **Build-Konfigurationen**. Alle vorhandenen Build-Konfigurationen sind hier aufgeführt.
2. Verwenden Sie eines der folgenden Verfahren:
 - Um eine neue Build-Konfiguration auf der Basis der aktuellen Einstellungen im Dialogfeld **Projekt>Optionen** zu erstellen, klicken Sie den Knoten Build-Konfigurationen mit der rechten Maustaste an und wählen **Neue hinzufügen**.
 - Zum Erstellen einer neuen Build-Konfiguration auf der Basis einer vorhandenen Build-Konfiguration klicken Sie den Namen der Konfiguration (wie z.B. **Debug** oder **Ausgabe**) mit der rechten Maustaste an und wählen **Neue hinzufügen**.

So ändern Sie eine vorhandene Build-Konfiguration:

1. Wählen Sie **Projekt>Optionen**. Build-Konfiguration ist das erste Feld auf allen Build-bezogenen Seiten.
 2. Wählen Sie auf einer beliebigen Seite, die das Feld Build-Konfiguration enthält, den Name der gewünschten Build-Konfiguration aus.
 3. Passen Sie die Projektoptionen auf allen Build-bezogenen Seiten entsprechend Ihren Bedürfnissen an.
- Tip:** Um eine Build-Konfiguration für ein Projekt oder eine Projektgruppe anzuwenden, wählen Sie **Projekt>Konfigurations-Manager** und klicken auf **Übernehmen**.

Siehe auch

Überblick zu MSBuild (siehe Seite 1461)

Überblick zu Build-Konfigurationen (siehe Seite 1463)

Manager für Build-Konfigurationen (siehe Seite 496)

Übernehmen der aktiven Build-Konfiguration (siehe Seite 2)

1.7 Ressourcendateien im Translation-Manager bearbeiten

Geben Sie einen Einleitungstext ein, falls dies erforderlich ist. Andernfalls beginnen Sie mit der Anleitung. Nehmen Sie keine grundlegenden Informationen zum Thema in die Anleitung auf. Bei komplexeren Anleitungen mit mehreren Phasen verwenden Sie die erste Aufgabenliste, um die erforderlichen Schritte zu skizzieren. Danach geben Sie für jede nachfolgende Phase Teilaufgaben an.

So bearbeiten Sie eine Ressourcendatei Translation-Manager:

1. Do this. When indicating a location in the IDE, provide the locator first, followed by the action. For example, "On the Component palette, click the Standard page..."
2. Do that. Use the appropriate XML tag for IDE elements such as dialog box names and fields.
3. Do this. Save this file in the HowTo directory, using the [TBD] naming convention.
4. If the procedure exceeds nine steps, consider splitting it into subtasks.

To drop another component on a form

1. Do this. Avoid more than three subtasks in a file. If documenting more than three subtasks, consider reorganizing into separate procedure files. Use a procedure task list to link procedure files together.
2. Do that.
3. Do the other.

Add optional summary information?

To drop yet another component on a form

1. Do this. Avoid more than three [TBD] subtasks in a file.
2. Do that.
3. Do the other.

Add optional summary information?

Tip: Use a tip to provide optional information, such as a shortcut key.

Anmerkung: Use a note to provide important information that might prevent the procedure from working.

Warnung: Use a warning to indicate a serious danger, such as loss of data.

1.8 Weitere Programmiersprachen installieren

Wenn Sie RAD Studio nur mit ein oder zwei Programmiersprachen (Delphi, C#, C++) installiert haben und später eine weitere, ursprünglich nicht installierte Sprache hinzufügen möchten, führen Sie die nachfolgend beschriebenen Schritte aus.

So fügen Sie der IDE weitere Programmiersprachen hinzu:

1. Wählen Sie **Start**▶**Einstellungen**▶**Systemsteuerung**▶**Software**.
2. Wählen Sie RAD Studio.
3. Klicken Sie auf die Schaltfläche Ändern.
4. Im angezeigten Installationsassistenten (von Windows) werden Sie gefragt, ob Sie das Programm ändern, reparieren oder entfernen möchten. Aktivieren Sie das Optionsfeld Ändern.
5. Führen Sie die restlichen Schritte im Installationsassistenten aus, und wählen Sie die Sprachen, die hinzugefügt werden sollen.
6. Klicken Sie auf die Schaltfläche Fertig stellen.

1.9 Delphi-Units in eine Anwendung linken

Wenn eine Anwendung kompiliert wird, die Referenzen auf eine von Delphi erstellte Assemblierung enthält, können die Delphi-Units für diese Assemblierung in die Anwendung gelinkt werden. Der Compiler linkt die binären DCUIL-Dateien hinzu, sodass die Assemblierung nicht mit der Anwendung weitergegeben werden muss.

So linken Sie eine Delphi-Unit hinzu:

1. Öffnen Sie die Anwendung in der IDE, und wählen Sie **Projekt**▸**Referenz hinzufügen**.
2. Wählen Sie im Dialogfeld Referenz hinzufügen in der Liste der .NET-Assemblierungen eine von Delphi generierte Assemblierungs-DLL aus, und klicken Sie auf die Schaltfläche Referenz hinzufügen. Wenn sich die gewünschte Assemblierung nicht in der Liste befindet, klicken Sie auf die Schaltfläche Durchsuchen, navigieren zu der Datei und wählen sie aus.
3. Klicken Sie auf OK. Die Assemblierung wird unter dem Knoten Referenzen in der Projektverwaltung aufgelistet.
4. Klicken Sie in der Projektverwaltung mit der rechten Maustaste auf die Assemblierung, und wählen Sie den Menübefehl Delphi-Units linken. Der Menübefehl ist inaktiv, wenn es sich nicht um eine von Delphi generierte Assemblierung handelt. Die zugehörige Eigenschaft *Link Units* wird im Objektinspektor auf **True** gesetzt.
5. Wählen Sie **Projekt**▸**Compilieren**, um die Anwendung zu compilieren.

1.10 Ein Symbol umbenennen

Sie können ein Symbol umbenennen, wenn sich das ursprüngliche Deklarationssymbol innerhalb Ihres Projekts oder in einem Projekt befindet, das von Ihrem Projekt abhängig ist und zur selben geöffneten Projektgruppe gehört. Es ist auch möglich, Fehlersymbole umzubenennen.

So benennen Sie ein Symbol um:

1. Wählen Sie den Symbolnamen im Quelltext-Editor aus.
2. Klicken Sie mit der rechten Maustaste, um das Kontextmenü anzuzeigen.
3. Wählen Sie **Refactoring**▶**'Symboltyp' umbenennen 'Symbolname'**, wobei für **Symboltyp** entweder Methode, Variable oder Feld stehen kann und **Symbolname** der Name des ausgewählten Symbols ist. Das Dialogfeld Umbenennen wird geöffnet.
4. Geben Sie den neuen Namen in das Feld **Neuer Name** ein.
5. Wenn Sie die Änderung an den Projektdateien in der Vorschau anzeigen möchten, aktivieren Sie das Kontrollkästchen **Vor Refactoring Referenzen anzeigen**.

Anmerkung: Die Menübefehle sind kontextsensitiv. Wenn eine Methode ausgewählt ist, wird **Methode umbenennen** *Methodenname* angezeigt, wobei *Methodenname* für den Namen der ausgewählten Methode steht. Dieses kontextsensitive Verhalten gilt auch für alle anderen Objekttypen.

Siehe auch

Überblick zum Refactoring (siehe Seite 1412)

1.11 Erzeugen eines Projekts mit einem MSBuild-Befehl

Die IDE verwendet zum Erzeugen eines Projekts die MSBuild-Engine von Microsoft. Sie können ohne Kenntnisse über MSBuild Projekte erstellen; die IDE handhabt alle Details für Sie. Projekte lassen sich aber auch direkt - wie im Folgenden beschrieben - über die Befehlszeilsyntax von MSBuild erzeugen. Wenn Sie ein Projekt erzeugen, erscheinen die Ergebnisse des Build-Prozesses auf der Registerkarte Ausgabe des Fensters Meldungen. Wenn Sie Build-Ereignisse festgelegt haben, zeigt der Build-Ausgabebereich die von Ihnen angegebenen Befehle und deren Ergebnisse an.

Die Befehlszeilsyntax von MSBuild hat die folgende Form:

```
MSBuild <projektname> [/t:<zielname>] [/p:configuration=<konfigurationsname>]
```

So erzeugen Sie ein Projekt über die Befehlszeile:

1. Wählen Sie im Startmenü **CodeGear RAD Studio** ▶ **RAD Studio-Befehlszeile**. Das Befehlszeilenfenster richtet automatisch die Umgebung für die Verwendung der RAD Studio-Tools, wie MSBuild.exe, ein.
2. Navigieren Sie zu dem Verzeichnis, das Ihr Projekt enthält, wie z.B. C:\Dokumente und Einstellungen\<meinname>\Eigene Dateien\RAD Studio\Projekte.
3. Geben Sie `msbuild` ein, aber drücken Sie die Taste **Eingabe** noch nicht.
4. Geben Sie den Namen Ihres Projekts, wie z.B. `TelePoll.dproj` (ein Delphi-Projekt) oder `User.Info.cbproj` (ein C++-Projekt), ein. Wenn sich das Projekt nicht im aktuellen Verzeichnis befindet, müssen Sie den vollständigen Pfadnamen des Projekts angeben.
5. Zum Festlegen eines Ziels geben Sie das Tag `/t:` gefolgt von einem in Ihrer Projektdatei festgelegten Ziel ein. Die drei Standardzielnamen sind `clean`, `make` und `build`:
 - `Clean` bedeutet, dass das Projekt bereinigt wird. Dabei werden erzeugte Dateien, wie z.B. Objektcode, entfernt. `Clean` entspricht dem Kontextmenübefehl Bereinigen in der Projektverwaltung.
 - `Make` bedeutet, dass das Projekt compiliert wird. `Make` entspricht dem Kontextmenübefehl Compilieren.
 - `Build` bedeutet, dass das Projekt erzeugt wird. `Build` entspricht dem Kontextmenübefehl Erzeugen. Die drei Ziele entsprechen den Befehlen **Bereinigen**, **Compilieren** und **Erzeugen** im Kontextmenü der Projektverwaltung. Das Standardziel ist `build`.
6. Zum Festlegen einer Konfiguration geben Sie den Namen der Konfiguration hinter `/p:configuration = ein`. Wenn Sie keine Konfiguration festlegen verwendet MSBuild die aktuell aktive Konfiguration. Verwenden Sie für die Konfiguration einen in Ihrem Projekt vorhandenen Konfigurationsnamen. Das kann eine Standardkonfiguration, wie z.B. **Debug**, oder eine Konfiguration sein, die Sie dem Projekt hinzugefügt haben. Wenn der Konfigurationsname ein Leerzeichen enthält, setzen Sie den Namen in doppelte Anführungszeichen, wie z.B.: `/p:configuration = "My config"`
7. Geben Sie weiter Optionen an, und drücken Sie **Eingabe**, um das Erzeugen zu starten.

Um die Online-Hilfe für MSBuild (mit einem ausführlichen Beispiel einer Befehlszeile) anzuzeigen, öffnen Sie die RAD Studio-Befehlszeile (siehe Schritt 1) und geben `MSBuild /help` ein.

Weitere Informationen über MSBuild finden Sie in der Microsoft-Dokumentation unter <http://msdn.microsoft.com>.

Siehe auch

Überblick zu MSBuild (siehe Seite 1461)

Compilieren (siehe Seite 1351)

IDE-Befehlszeilenoptionen (siehe Seite 1333)

Manager für Build-Konfigurationen (siehe Seite 496)

Erstellen von benannten Build-Konfigurationen

Von RAD Studio erzeugte Dateien (siehe Seite 384)

1.12 Verwenden von Targets-Dateien

So erstellen Sie im Projekt über das Menü eine neue .targets-Datei:

1. Wählen Sie **Datei > Neu > Weitere**.
2. Klicken Sie unter C++Builder-Projekte auf C++Builder-Dateien.
3. Wählen Sie links MSBuild Targets-Datei.
4. Eine neue .targets-XML-Datei wird erstellt und im IDE-Fenster angezeigt. Sie enthält nur einen **<Project>**-Knoten.

So fügen Sie dem Projekt über das Menü eine .targets-Datei hinzu:

1. Wählen Sie **Projekt > Dem Projekt hinzufügen**.
2. Wählen Sie aus der Pulldown-Liste Dateityp des Dialogfeldes Dem Projekt hinzufügen den Eintrag MSBuild targets-Datei (*.targets) aus.
3. Navigieren Sie zu der .targets-Datei.
4. Klicken Sie auf Öffnen, um die Datei dem Projekt hinzuzufügen und das Dialogfeld zu schließen. Klicken Sie auf Abbrechen, um die Datei nicht hinzuzufügen und das Dialogfeld zu schließen.

So fügen Sie dem Projekt über die Projektverwaltung eine .targets-Datei hinzu:

1. Klicken Sie das Projekt in der Projektverwaltung mit der rechten Maustaste an.
2. Wählen Sie im Kontextmenü Hinzufügen....
3. Wählen Sie aus der Pulldown-Liste Dateityp des Dialogfeldes Dem Projekt hinzufügen den Eintrag MSBuild targets-Datei (*.targets) aus.
4. Navigieren Sie zu der .targets-Datei.
5. Klicken Sie auf Öffnen, um die Datei dem Projekt hinzuzufügen und das Dialogfeld zu schließen. Klicken Sie auf Abbrechen, um die Datei nicht hinzuzufügen und das Dialogfeld zu schließen.

So aktivieren Sie eine .targets-Datei:

1. Klicken Sie die .targets-Datei in der Projektverwaltung mit der rechten Maustaste an.
2. Wählen Sie im Kontextmenü Aktivieren.

So überprüfen Sie, ob eine .targets-Datei passend und fehlerfrei ist:

1. Klicken Sie die .targets-Datei in der Projektverwaltung mit der rechten Maustaste an.
2. Wählen Sie im Kontextmenü Überprüfen.

So entfernen Sie eine .targets-Datei aus dem Projekt:

1. Klicken Sie die .targets-Datei in der Projektverwaltung mit der rechten Maustaste an.
2. Wählen Sie im Kontextmenü Aus dem Projekt entfernen.

Siehe auch

Targets-Dateien (siehe Seite 1471)

1.13 Arbeiten mit benannten Optionsgruppen

So erstellen Sie eine benannte Optionsgruppe über das Dialogfeld Projektoptionen:

1. Wählen Sie **Projekt > Optionen**.
2. Build-Konfiguration ist das erste auf der Seite. Wählen Sie aus der Dropdown-Liste die Konfiguration aus, auf der Ihre benannte Optionsgruppe basieren soll.
3. Passen Sie die Optionen auf den Seiten an, die den Build-Konfigurationen zugeordnet sind.
4. Klicken Sie auf die Schaltfläche Speichern unter....
5. Geben Sie im Dialogfeld Speichern unter einen Namen für die Optionsgruppe ein.
6. Klicken Sie auf OK, um die Optionsgruppe in eine Datei zu speichern.

So erstellen Sie eine benannte Optionsgruppe in der Projektverwaltung:

1. Klicken Sie in der Projektverwaltung die Konfiguration mit der rechten Maustaste an, aus der Sie eine Optionsgruppe speichern möchten.
2. Klicken Sie im Kontextmenü auf Speichern unter....
3. Geben Sie im Dialogfeld Speichern unter einen Namen für die Optionsgruppe ein.
4. Klicken Sie auf OK, um die Optionsgruppe in eine Datei zu speichern.

So laden Sie eine benannte Optionsgruppe über das Dialogfeld Projektoptionen:

1. Wählen Sie **Projekt > Optionen**.
2. Wählen Sie aus der Drop-Down-Liste den Namen der Build-Konfiguration aus, für die Sie die Optionsgruppe übernehmen möchten.
3. Klicken Sie auf Laden..., um das Dialogfeld Optionsgruppe anzuzeigen.
4. Navigieren Sie zu der gewünschten Datei mit der benannten Optionsgruppe.
5. Wählen Sie, wie die Werte aus der benannten Optionsgruppe geladen werden sollen: Überschreiben, Ersetzen oder Beibehalten.
6. Klicken Sie auf OK, um die Optionsgruppe zu übernehmen und das Dialogfeld zu schließen. Klicken Sie auf Abbrechen, um die Optionsgruppe nicht zu übernehmen und das Dialogfeld zu schließen.

So laden Sie eine benannte Optionsgruppe in der Projektverwaltung:

1. Klicken Sie in der Projektverwaltung die Konfiguration mit der rechten Maustaste an, für die Sie eine Optionsgruppe übernehmen möchten.
2. Klicken Sie im Kontextmenü auf Optionsgruppe anwenden..., um das Dialogfeld Optionsgruppe anzuzeigen.
3. Navigieren Sie zu der gewünschten Datei mit der benannten Optionsgruppe.
4. Wählen Sie, wie die Werte aus der benannten Optionsgruppe geladen werden sollen: Überschreiben, Ersetzen oder Beibehalten.
5. Klicken Sie auf OK, um die Optionsgruppe zu übernehmen und das Dialogfeld zu schließen. Klicken Sie auf Abbrechen, um die Optionsgruppe nicht zu übernehmen und das Dialogfeld zu schließen.

Siehe auch

Optionsgruppe anwenden (Dialogfeld) ( siehe Seite 492)

Überblick zu MSBuild ( siehe Seite 1461)

Überblick zu C++Build-Konfigurationen (siehe Seite 1465)

Manager für Build-Konfigurationen (siehe Seite 496)

Festlegen von C++-Standardprojektoptionen (siehe Seite 76)

Erstellen von benannten Build-Konfigurationen (C++) (siehe Seite 6)

Übernehmen der aktiven Build-Konfiguration (siehe Seite 2)

1.14 Ausdruck hinzufügen

Mit einem überwachten Ausdruck können Sie die Werte von Programmvariablen oder Ausdrücken überprüfen, während Sie Ihren Quelltext routinen- oder schrittweise debuggen. Bei jeder Unterbrechung der Programmausführung wertet der Debugger alle Elemente auf der Registerkarte **Active** (bzw. **ActiveWatchGroup**) im Fenster Liste überwachter Ausdrücke aus und aktualisiert die angezeigten Werte.

Sie können überwachte Ausdrücke auch in Gruppen zusammenfassen. Wenn Sie eine Gruppe überwachter Ausdrücke einfügen, wird im Fenster Liste überwachter Ausdrücke eine neue Registerkarte erstellt. Wird ein Gruppenregister angezeigt, werden während des Debuggens nur die überwachten Ausdrücke in dieser Gruppe ausgewertet. Durch die Gruppierung können Sie auch verhindern, dass überwachte Ausdrücke, die außerhalb des Gültigkeitsbereichs liegen, das schrittweise Debuggen verlangsamen.

So fügen Sie einen überwachten Ausdruck hinzu:

1. Wählen Sie **Start > Ausdruck hinzufügen**, um das Dialogfeld Darstellung überwachter Ausdrücke anzuzeigen.
2. Geben Sie in das Feld Ausdruck den Ausdruck ein, den Sie überwachen möchten. Ein Ausdruck besteht aus Konstanten, Variablen und Werten in Datenstrukturen sowie den Operatoren. Nahezu jeder Ausdruck, der auf der rechten Seite eines Zuweisungsoperators verwendet wird, kann auch als Debug-Ausdruck eingesetzt werden. Eine Ausnahme bilden Variablen, auf die zum aktuellen Zeitpunkt der Programmausführung kein Zugriff besteht.
3. Möchten Sie den überwachten Ausdruck in einer neuen Gruppe erstellen, geben Sie in das Feld Gruppenname einen neuen Namen ein, oder wählen Sie einen bereits definierten Gruppennamen aus der Liste aus.
4. Stellen Sie die weiteren Optionen nach Bedarf ein (klicken Sie im Dialogfeld Darstellung überwachter Ausdrücke auf Hilfe, um eine Erläuterung zu diesen Optionen einzublenden). Sie können den Debugger auch dann zur Auswertung überwachter Ausdrücke einsetzen, wenn dies zu Funktionsaufrufen führt, indem Sie die Option Funktionsaufrufe gestatten auswählen.
5. Klicken Sie auf OK.

Der überwachte Ausdruck wird in das Fenster Liste überwachter Ausdrücke eingefügt.

Siehe auch

[Darstellung überwachter Ausdrücke \(siehe Seite 672\)](#)

[Liste überwachter Ausdrücke \(Fenster\) \(siehe Seite 802\)](#)

1.15 Mit einem ausgeführten Prozess verbinden

Sie können eine Verbindung zu einem Prozess herstellen, der auf Ihrem oder einem externen Computer ausgeführt wird. Dies ist sinnvoll, wenn ein Programm auf Fehler geprüft werden soll, das nicht in RAD Studio erstellt wurde.

So stellen Sie eine Verbindung mit einem ausgeführten Prozess her:

1. Wählen Sie **Start > Mit Prozess verbinden**, um das Dialogfeld Mit Prozess verbinden anzuzeigen.
2. Wählen Sie in der Dropdown-Liste Debugger entweder CodeGear .NET Debugger oder CodeGear Win32 Debugger aus. Ihre Auswahl hängt davon ab, ob Sie eine Verbindung zu einem .NET- oder einem Win32-Prozess herstellen möchten. Die Liste Ausgeführte Prozesse wird aktualisiert und zeigt die entsprechenden Prozesse an. Für Win32-Prozesse können Sie auch die Option Systemprozesse anzeigen aktivieren, damit die Systemprozesse in die Liste aufgenommen werden.
3. Wenn der Prozess auf einem externen Computer ausgeführt wird, geben Sie den Namen des Computers in das Feld Externer Rechner ein.

Anmerkung: Der externe Debug-Server muss auf dem externen Computer laufen.

4. Wählen Sie einen Prozess in der Liste Ausgeführte Prozesse aus.
5. Wenn der Prozess nach dem Herstellen der Verbindung nicht angehalten werden soll, deaktivieren Sie das Kontrollkästchen Pause nach Verbindung.
6. Klicken Sie auf Verbinden.

Siehe auch

Mit Prozess verbinden ( siehe Seite 656)

1.16 Quelltexthaltepunkte setzen und bearbeiten

Haltepunkte dienen dazu, die Programmausführung an einer bestimmten Stelle oder beim Auftreten einer bestimmten Bedingung zu unterbrechen. Die Haltepunkte lassen sich vor und während einer Debug-Sitzung im Quelltext-Editor setzen. Während einer Debug-Sitzung wird jede Quelltextzeile, die für einen Haltepunkt in Frage kommt, mit einem blauen Punkt • in der Leiste am linken Rand des Quelltext-Editors gekennzeichnet.

Sie können Haltepunkte auch für Frames setzen, die im Fenster Aufruf-Stack angezeigt werden. Die Haltepunktsymbole im Fenster Aufruf-Stack entsprechen denen im Quelltext-Editor, außer dass der blaue Punkt • nur angibt, dass Debug-Informationen für den Frame zur Verfügung stehen und nicht ob ein Haltepunkt für den Frame gesetzt werden kann.

So setzen Sie einen Haltepunkt:

1. Klicken Sie im Quelltext-Editor neben der Zeile, an der die Programmausführung unterbrochen werden soll, in die Leiste am linken Rand.
2. Wählen Sie **Start**▸**Haltepunkt hinzufügen**▸**Quelltexthaltepunkt**, um das Dialogfeld Quelltexthaltepunkt hinzufügen anzuzeigen.

Tip: Um die Leiste im Quelltext-Editor zu vergrößern, wählen Sie **Tools**▸**Optionen**▸**Editor-Optionen**▸**Anzeige** und erhöhen den Wert im Feld Leistenbreite.

3. Geben Sie die gewünschten Werte ein, und klicken Sie auf OK.

Die folgenden Symbole werden zur Darstellung von Haltepunkten in der Leiste des Quelltext-Editors verwendet.

Symbol	Beschreibung
•	Der Haltepunkt ist gültig und aktiviert. Der Debugger ist nicht aktiv.
• (grün)	Der Haltepunkt ist gültig und aktiviert. Der Debugger ist aktiv.
• (rot)	Der Haltepunkt ist ungültig und aktiviert. Der Haltepunkt wurde an einer ungültigen Position gesetzt, zum Beispiel in einem Kommentar, einer Leerzeile oder einer ungültigen Deklaration.
○	Der Haltepunkt ist gültig und deaktiviert. Der Debugger ist nicht aktiv.
○ (grün)	Der Haltepunkt ist gültig und deaktiviert. Der Debugger ist aktiv.
○ (rot)	Der Haltepunkt ist ungültig und deaktiviert. Der Haltepunkt wurde an einer ungültigen Position gesetzt.

Haltepunkte werden im Fenster Liste der Haltepunkte angezeigt.

So bearbeiten Sie einen Haltepunkt:

1. Klicken Sie mit der rechten Maustaste auf das Symbol des Haltepunkts, und wählen Sie Eigenschaft des Haltepunkts.
2. Stellen Sie die Optionen im Dialogfeld Eigenschaft des Haltepunkts nach Bedarf ein, um den Haltepunkt zu bearbeiten. Sie können zum Beispiel eine Bedingung angeben, eine Haltepunktgruppe definieren oder festlegen, welche Aktion erfolgen soll, wenn die Ausführung an diesem Haltepunkt angelangt ist.
3. Klicken Sie auf Hilfe, um weitere Informationen zu den Optionen in diesem Dialogfeld zu erhalten.
4. Klicken Sie auf OK.

So erstellen Sie eine Haltepunktgruppe:

1. Klicken Sie mit der rechten Maustaste auf das Symbol des Haltepunkts, und wählen Sie Eigenschaft des Haltepunkts.

2. Geben Sie in das Feld Gruppe einen Namen für die Gruppe ein, oder wählen Sie im Listenfeld den Namen der gewünschten Gruppe aus.
3. Klicken Sie auf OK.

So aktivieren oder deaktivieren Sie einen Haltepunkt bzw. eine Haltepunktgruppe:

1. Klicken Sie mit der rechten Maustaste im Quelltext-Editor oder im Fenster Liste der Haltepunkte auf das Haltepunktsymbol, und wählen Sie Aktiviert, um zwischen Aktivierung und Deaktivierung umzuschalten.
2. Sollen alle Haltepunkte aktiviert oder deaktiviert werden, klicken Sie mit der rechten Maustaste auf einen leeren Bereich (keinen Haltepunkt) im Fenster Liste der Haltepunkte und wählen Alle einschalten oder Alle ausschalten.
3. Um eine Gruppe von Haltepunkten zu aktivieren oder zu deaktivieren, klicken Sie mit der rechten Maustaste im Fenster Liste der Haltepunkte in einen leeren Bereich (nicht auf einen Haltepunkt), und wählen Sie Gruppe aktivieren oder Gruppe deaktivieren.

Tip: Wenn Sie die Taste **STRG** gedrückt halten und im Quelltext-Editor auf einen Haltepunkt klicken, wird zwischen der Aktivierung und Deaktivierung des Haltepunkts umgeschaltet.

Wenn Sie einen Haltepunkt oder eine Haltepunktgruppe deaktivieren, verhindern Sie damit lediglich, dass die Programmausführung in dieser Quelltextzeile unterbrochen wird. Die Haltepunkte und deren Einstellungen bleiben aber erhalten und lassen sich später jederzeit wieder aktivieren.

So erstellen Sie einen bedingten Haltepunkt:

1. Wählen Sie **Start**▶**Haltepunkt hinzufügen**▶**Quelltexthaltepunkt**, um das Dialogfeld Quelltexthaltepunkt hinzufügen anzuzeigen.
2. Geben Sie in das Feld Zeilennummer die Nummer der Zeile im Quelltext-Editor ein, in der der Haltepunkt gesetzt werden soll.
- Tip:** Die entsprechende Zeilennummer ist bereits in das Feld Zeilennummer eingetragen, wenn Sie im Quelltext-Editor auf eine Zeile klicken, bevor Sie das Dialogfeld Quelltexthaltepunkt hinzufügen öffnen.
3. Geben Sie in das Feld Bedingung einen Bedingungsausdruck ein, der jedes Mal ausgewertet werden soll, wenn die Programmausführung an diesen Haltepunkt gelangt.
4. Klicken Sie auf OK.

Mit Hilfe von bedingten Haltepunkten können Sie untersuchen, wie sich Ihr Programm verhält, wenn eine Variable einen bestimmten Wert annimmt oder ein bestimmtes Flag gesetzt ist.

Ergibt die Auswertung des Bedingungsausdrucks den Wert **True** (oder nicht Null), unterbricht der Debugger das Programm an der Position dieses Haltepunkts. Ergibt die Auswertung des Ausdrucks **False** (oder Null), unterbricht der Debugger die Ausführung an dieser Haltepunktposition nicht.

So können Sie einem Haltepunkt Aktionen zuweisen:

1. Wählen Sie **Start**▶**Haltepunkt hinzufügen**▶**Quelltexthaltepunkt**, um das Dialogfeld Quelltexthaltepunkt hinzufügen anzuzeigen.
- Tip:** Alternativ klicken Sie mit der rechten Maustaste auf das Haltepunktsymbol und wählen Eigenschaft des Haltepunkts, um das Dialogfeld Eigenschaft des Haltepunkts anzuzeigen.
2. Klicken Sie auf Weitere, um zusätzliche Optionen am unteren Rand des Dialogfelds anzuzeigen.
3. Aktivieren Sie jene Aktionen, die erfolgen sollen, wenn die Ausführung an dem betreffenden Haltepunkt angelangt ist. Sie können beispielsweise einen Ausdruck angeben, der ausgewertet werden soll. Das Ergebnis der Auswertung wird in das Ereignisprotokoll übernommen.
4. Klicken Sie auf OK.

So ändern Sie die Farbe der Haltepunkte und des Textes an der Ausführungsposition:

1. Wählen Sie **Tools**▶**Optionen**▶**Editor-Optionen**▶**Farbe**.
2. Wählen Sie im Beispiel-Quelltextfenster das Register der betreffenden Programmiersprache aus. Um beispielsweise die Haltepunktfarbe für RAD Studio-Quelltext zu ändern, aktivieren Sie das Register RAD Studio.
3. Verschieben Sie den Ausschnitt im Beispiel-Quelltextfenster nach rechts, um die Symbole für die Ausführungsposition und die Haltepunkte in der linken Leiste des Fensters anzuzeigen.
4. Klicken Sie an einer beliebigen Position auf die Symbole für die Ausführungsposition, die Haltepunkte oder die Zeile, die Sie ändern möchten.
5. Verwenden Sie die Listenfelder Vordergrundfarbe und Hintergrundfarbe, um die Farben für die Ausführungsposition oder den Haltepunkt zu ändern.
6. Klicken Sie auf OK.

Anmerkung: Haltepunkte können auch in der Liste der Haltepunkte, in dem Fenster CPU (und in der Disassembly-Ansicht), in der Ansicht Aufruf-Stack und in dem Fenster Module gesetzt werden.

Siehe auch

Quelltexthaltepunkt hinzufügen/Adresshaltepunkt hinzufügen (Dialogfeld) (↗ siehe Seite 1349)

1.17 VCL für .NET-Quelltext debuggen

Um VCL für .NET-Quelltext debuggen zu können, müssen Sie bestimmte Projektoptionen festlegen, die für das Debuggen anderer Anwendungsarten nicht erforderlich sind. Diese Optionen sind standardmäßig deaktiviert und müssen explizit festgelegt werden.

So aktivieren Sie Optionen für das Debuggen von VCL für .NET-Quelltext:

1. Öffnen Sie ein VCL für .NET-Projekt.
2. Wählen Sie **Projekt**▶**Optionen**▶**Compiler**.
3. Aktivieren Sie das Kontrollkästchen **Mit Debug-DCUILs**.
4. Klicken Sie auf **OK**.
5. Wählen Sie in der Projektverwaltung unter **Referenzen** eine Borland-Assemblierung aus.
6. Klicken Sie mit der rechten Maustaste auf die Assemblierung, und wählen Sie **In Delphi-Units verknüpfen**. Dadurch wird die Eigenschaft **Link Units** im Objektinspektor auf **True** gesetzt.
7. Wiederholen Sie die beiden vorherigen Schritte für jede CodeGear-Assemblierung, die Sie debuggen möchten.

Nun können Sie den VCL für .NET-Quelltext debuggen.

Tip: Mit diesem Vorgehen lassen sich auch VCL für .NET-Assemblierungen debuggen, die von einem anderen Hersteller stammen. Allerdings müssen hierfür die entsprechenden Debug-DCUILs für die Assemblierungen zur Verfügung stehen.

Siehe auch

[Delphi-Units in eine Anwendung linken](#) (siehe Seite 10)

[.NET-Assemblierungen](#) (siehe Seite 475)

[Compiler](#) (siehe Seite 501)

[Projektverwaltung](#) (siehe Seite 810)

1.18 Kurzhinweise während des Debuggens verwenden

Beim Debuggen einer Anwendung können Sie die Werte von Membern eines überwachten Objekts anzeigen, wenn es sich um ein komplexes Datenobjekt (wie eine Klasse, einen Record oder ein Array) handelt. Die Werte werden im Quelltext-Editor angezeigt, wenn Sie ein überwachtes Objekt erweitern. Außerdem haben Sie die Möglichkeit, die Elemente innerhalb eines Objekts zu erweitern und deren Unterelemente und Werte anzuzeigen. Sie können alle Objektebenen erweitern. Member werden nach Vorfahr gruppierter.

So erweitern Sie Kurzhinweise während des Debuggens:

1. Erstellen Sie eine neue VCL für Win32-Anwendung, oder öffnen Sie eine vorhandene Anwendung.
2. Wählen Sie **Projekt**▸**Optionen**▸**Compiler**, und vergewissern Sie sich, dass die Option Mit Debug-DCUs aktiviert ist.
3. Wählen Sie **Tools**▸**Optionen**▸**Editor-Optionen**▸**Programmierhilfe**, und stellen Sie sicher, dass die Option Auswertung durch Kurzhinweis aktiviert ist.
4. Wählen Sie **Start**▸**Gesamte Routine**.

Tip: Drücken Sie alternativ F8.

Die Seite Code der Hauptquelltextdatei für das Projekt wird geöffnet.

5. Wählen Sie erneut **Start**▸**Gesamte Routine**. Daraufhin wird das Projekt initialisiert.
6. Zeigen Sie mit der Maus auf das Schlüsselwort **Application**. Der Kurzhinweis wird in einem einzelnen Block angezeigt.
7. Klicken Sie auf das Pluszeichen (+) neben dem Schlüsselwort **Application** im Kurzhinweis. Der Kurzhinweis wird zu einem bildlauffähigen Feld vergrößert, in dem alle untergeordneten Eigenschaften und ihre Werte enthalten sind. Neben jeder Eigenschaft, die eine oder mehrere untergeordnete Eigenschaften besitzt, wird ein Pluszeichen (+) angezeigt. Sie können jedes Element erweitern und seine Eigenschaften und Werte im Kurzhinweis in hierarchischer Form anzeigen.

Siehe auch

Anwendungen debuggen (☞ siehe Seite 1439)

Compiler (☞ siehe Seite 501)

Programmierhilfe (☞ siehe Seite 728)

1.19 Werte von Datenelementen untersuchen und ändern

Im Debug-Inspektor können Sie Datenelemente untersuchen, da die angezeigten Datentypen automatisch formatiert werden. Der Debug-Inspektor ist sinnvoll, um zusammengesetzte Datenobjekte wie Arrays und verkettete Listen zu überprüfen. Da Sie die verschiedenen im Debug-Inspektor angezeigten Datenelemente untersuchen können, können Sie das gesamte komplexe Datenobjekt *durchgehen*, indem Sie für die Komponenten des Objekts jeweils einen Debug-Inspektor öffnen.

Anmerkung: Der Debug-Inspektor ist nur verfügbar, wenn ein Prozess im Debugger angehalten wurde.

So können Sie ein Datenelement direkt im Quelltext-Editor untersuchen:

1. Setzen Sie die Einfügemarkierung im Quelltext-Editor auf jenes Datenelement, das Sie überprüfen möchten.
2. Klicken Sie mit der rechten Maustaste, und wählen Sie **Fehlersuche**▶**Untersuchen**, um den Debug-Inspektor anzuzeigen.

So können Sie ein Datenelement über das Menü untersuchen:

1. Wählen Sie **Start**▶**Untersuchen**, um das Dialogfeld Untersuchen anzuzeigen.
2. Im Dialogfeld Untersuchen geben Sie den Ausdruck ein, den Sie überprüfen möchten.
3. Klicken Sie auf OK. Der Debug-Inspektor wird angezeigt.

Anders als bei überwachten Ausdrücken, ist der Gültigkeitsbereich von Datenelementen im Debug-Inspektor auf die Zeit beschränkt, in der die Auswertung stattfindet. Wenn Sie den Befehl Untersuchen aus dem Quelltext-Editor wählen, verwendet der Debugger die Position der Einfügemarkierung dazu, den Gültigkeitsbereich des untersuchten Ausdrucks zu definieren. So können Datenelemente untersucht werden, die sich nicht im aktuellen Gültigkeitsbereich des Ausführungspunkts befinden.

Wenn Sie den Befehl **Start**▶**Untersuchen** verwenden, wird das Datenelement im Gültigkeitsbereich des Ausführungspunkts ausgewertet.

Wenn der Ausführungspunkt innerhalb des Gültigkeitsbereichs des Ausdrucks liegt, den Sie untersuchen, wird der Wert im Debug-Inspektor angezeigt. Liegt der Ausführungspunkt außerhalb des Gültigkeitsbereichs des Ausdrucks, ist der Wert nicht definiert und der Debug-Inspektor bleibt leer.

So zeigen Sie Member des untersuchten Objekts an:

1. Klicken Sie auf die Registerkarte Daten, um Strings, boolesche Werte und andere Werte für Variablennamen, Ausdruck oder übergeordnete Komponente anzuzeigen.
2. Klicken Sie auf die Registerkarte Methoden, um alle Methoden anzuzeigen, die Member der Klasse des Objekts sind.
3. Klicken Sie auf das Register Eigenschaften, um sämtliche Eigenschaften des aktiven Objekts anzuzeigen.
4. Wenn Sie auf den Namen einer Eigenschaft klicken, wird deren Typ in der Statusleiste des Debug-Inspektors angezeigt.
5. Klicken Sie auf das Symbol mit dem Fragezeichen (?), um den aktuellen Wert dieser Eigenschaft zum aktuellen Ausführungspunkt der Anwendung einzublenden.

So ändern Sie den Wert eines Datenelements:

1. Wählen Sie im Debug-Inspektor ein Datenelement aus, das mit einer Ellipse (...) versehen ist. Sie erkennen daran, dass sich das Datenelement ändern lässt.
2. Klicken Sie auf die Ellipse (...) oder mit der rechten Maustaste auf das Element und wählen Sie Ändern.
3. Geben Sie einen neuen Wert ein, und klicken Sie auf OK.

So untersuchen Sie die Werte lokaler Variablen:

1. Solange die Anwendung im Debug-Modus ausgeführt wird, doppelklicken Sie auf irgendeine der Variablen, die im Fenster Lokale Variablen angezeigt werden. Daraufhin wird der Debug-Inspektor für die betreffende lokale Variable angezeigt.

1.20 Variablenausdrücke bearbeiten

Nachdem Sie eine Variable oder ein Element einer Datenstruktur ausgewertet haben, können Sie den Wert ändern. Wird ein Wert mit dem Debugger geändert, tritt diese Änderung nur bei der Programmausführung in Kraft. Änderungen, die Sie im Dialogfeld Auswerten/Ändern vornehmen, haben keine Auswirkungen auf den Quelltext oder das compilierte Programm. Um diese Änderungen dauerhaft zu übernehmen, müssen Sie den Quelltext im Quelltext-Editor entsprechend bearbeiten und das Programm dann neu compilieren.

So ändern Sie den Wert eines Ausdrucks:

1. Wählen Sie **Start > Auswerten/Ändern**.
2. Geben Sie den Ausdruck in das Eingabefeld Ausdruck ein. Soll eine Komponenteneigenschaft geändert werden, geben Sie den Namen der Eigenschaft an, zum Beispiel `this.button1.Height` oder `Self.button1.Height`.
3. Geben Sie einen Wert in das Eingabefeld Neuer Wert ein. Dieser Ausdruck muss ein Ergebnis haben, das bezüglich der Zuweisung mit der Variable kompatibel ist, der Sie ihn zuweisen möchten. Führt die Zuweisung zu einem Compiler- oder Laufzeitfehler, handelt es sich in den meisten Fällen um einen ungültigen Änderungswert.
4. Wählen Sie Ändern. Der neue Wert wird im Feld Ergebnis angezeigt. Die Änderung einer Variablen kann nicht rückgängig gemacht werden, nachdem Sie auf Ändern geklickt haben. Um einen Wert wiederherzustellen, können Sie jedoch wieder den vorherigen Wert in das Feld Ausdruck eingeben und den Ausdruck erneut entsprechend ändern.

Anmerkung: Sie können einzelne Variablen oder Elemente in Arrays und Datenstrukturen ändern. Mit einem einzelnen Ausdruck kann jedoch nicht der gesamte Inhalt eines Arrays oder einer Datenstruktur geändert werden.

Warnung: Das Ändern von Werten (insbesondere von Zeigerwerten und Array-Indizes) kann unerwünschte Effekte nach sich ziehen, da Sie andere Variablen und Datenstrukturen überschreiben können. Seien Sie vorsichtig beim Ändern von Programmwerten aus dem Debugger heraus.

1.21 Das Projekt zum Debuggen vorbereiten

Die meisten Debugger-Optionen werden standardmäßig eingestellt. Sie können diese Optionen aber wie im Folgenden beschrieben anzeigen und ändern. Es stehen sowohl allgemeine IDE-Optionen als auch projektspezifische Optionen zur Verfügung. Die projektspezifischen Optionen richten sich nach dem Typ des aktiven Projekts, beispielsweise Delphi, Delphi .NET oder C#.

So aktivieren Sie den integrierten Debugger:

1. Wählen Sie **Tools**▶**Optionen**▶**Debugger-Optionen**.
2. Wählen Sie die Option Integriertes Debuggen aus.
3. Klicken Sie auf OK.
4. Zusätzlich können Sie auch die Einstellungen auf den weiteren Debugger-Seiten prüfen.

So stellen Sie Debugger-Optionen ein:

1. Wählen Sie **Projekt**▶**Optionen**.
2. Überprüfen Sie die Debugger-Optionen auf den Seiten des Dialogfelds Projektoptionen. Sehen Sie sich insbesondere folgende Seiten genauer an: Compiler, Linker, Verzeichnisse/Bedingungen, Versionsinfo und Debugger. Nicht jede Seite steht für alle Projekttypen zur Verfügung. So wird die Seite Versionsinfo beispielsweise nur für Win32-Projekte in Delphi angezeigt.
3. Klicken Sie auf OK.

Siehe auch

Debugger-Optionen (☞ siehe Seite 510)

1.22 Installieren, Starten und Anhalten des externen Debug-Servers

Mit dem externen Debugger können Sie eine RAD Studio-Anwendung debuggen, die sich auf einem externen Computer befindet. Sobald der externe Debug-Server auf dem externen Computer ausgeführt wird, können Sie über RAD Studio die Verbindung zu diesem Computer herstellen und mit dem Debuggen beginnen.

Voraussetzungen und Sicherheitserwägungen für das externe Debuggen

- Der lokale und der externe Rechner müssen über TCP/IP verbunden sein.
- Alle Dateien, die zum Debuggen der Anwendung erforderlich sind, müssen sich auf dem externen Rechner befinden, bevor Sie mit dem Debuggen beginnen. Dazu gehören ausführbare Dateien, DLLs, Assemblierungen, Datendateien und PDB-(Debug)-Dateien.
- Zusätzlich zu dem Port, über den der externe Debug-Server empfängt, wird für jede Anwendung, für die das Debugging ausgeführt wird, eine Verbindung geöffnet. Zusätzliche Port-Nummer werden von Windows dynamisch ausgewählt. Eine Firewall, die nur Verbindungen zu dem Empfangs-Port zulässt, verhindert die Ausführung des externen Debuggers.

Warnung: Die Verbindung zwischen RAD Studio und dem externen Debug-Server wird über einen einfachen TCP/IP-Socket hergestellt, der weder Verschlüsselung noch Authentifizierung unterstützt. Aus diesem Grund sollte der externe Debug-Server nicht auf einem Computer ausgeführt werden, auf den nicht-autorisierte Clients über ein Netzwerk zugreifen können.

So installieren und starten Sie den externen Debug-Server:

- Wenn RAD Studio auf einem externen Computer installiert ist, überspringen Sie Schritt 4. In diesem Fall steht der externe Debug-Server (`rmtdbg105.exe`) bereits zur Verfügung, standardmäßig im Verzeichnis `C:\Programme\CodeGear\RAD Studio\5.0\Bin`.
- Kopieren Sie die Datei `rmtdbg105.exe` aus dem Verzeichnis `RAD Studio\bin` auf Ihrem lokalen Computer in das gewünschte Verzeichnis auf dem externen Computer.. Wenn Sie eine verwaltete Anwendung debuggen, kopieren Sie auch die Datei `dbkpro105.dll`
- Für das Debuggen einer verwalteten Anwendung registrieren Sie die Datei `dbkpro105.dll` beim externen Computer, indem Sie das Registrierungsprogramm `regsvr32.exe` ausführen. Unter Windows XP geben Sie beispielsweise an der Eingabeaufforderung `C:\Windows\System32\regsvr32.exe dbkdebugproide100.dll` ein.
- Führen Sie auf dem externen Computer `rmtdbg105.exe` mit der folgenden Syntax aus: `rmtdbg105.exe [-listen [hostname:]port]` Die Parameter haben folgende Bedeutung:
 - `hostname` ist ein optionaler Host-Name oder eine TCP/IP-Adresse für die Anbindung an einen bestimmten Host, zum Beispiel `somehost` oder `127.0.0.1`. Wenn Sie `hostname` festlegen, müssen Sie auch `:port` angeben.
 - `port` ist eine optionale (erforderlich wenn `hostname` angegeben wurde) Port-Nummer oder ein Standardprotokollname, zum Beispiel `8000` oder `ftp`. Wenn die Port-Nummer weggelassen wird, wird die Nummer `64447` verwendet. Beispiele:
 - `rmtdbg105.exe`
 - `rmtdbg105.exe -listen 8000`
 - `rmtdbg105.exe -listen somehost:8000`
 - `rmtdbg105.exe -listen 127.0.0.1:8000`

Nachdem der externe Debug-Server gestartet wurde, erscheint sein Symbol  in der Windows-Task-Leiste.

So fahren Sie den externen Debug-Server herunter:

- Klicken Sie auf dem externen Computer in der Windows-Task-Leiste mit der rechten Maustaste auf das Symbol  CodeGear

Remote Debugger Listener.

2. Klicken Sie im Menü der Verknüpfung auf Beenden.

Das Herunterfahren des externen Debug-Servers wirkt sich nicht auf aktive Debug-Sitzungen aus.

Siehe auch

Überblick zum externen Debugger (siehe Seite 1441)

Verbindung für das externe Debuggen einrichten

Dateien für das externe Debuggen vorbereiten (siehe Seite 35)

1.23 Debugger auf einem externen Computer installieren

Um ein Projekt auf einem externen Computer zu debuggen, auf dem RAD Studio nicht installiert ist, müssen Sie die ausführbaren Dateien des externen Debuggers installieren. Sie können diese Dateien entweder direkt vom Installationsdatenträger aus installieren oder sie von einem Computer kopieren, auf dem RAD Studio installiert ist.

So installieren Sie den externen Debugger:

1. Verwenden Sie den Installationsdatenträger, sofern dieser zur Verfügung steht.
2. Ist dies nicht der Fall, verwenden Sie die Dateien des Computers, auf dem die IDE installiert ist.

So installieren Sie den externen Debugger vom Installationsdatenträger:

1. Legen Sie den Installationsdatenträger in den externen Computer ein.
2. Wählen Sie den Befehl zum Installieren des externen Debuggers.
3. Folgen Sie den Anweisungen des Experten.

So installieren Sie den externen Debugger, wenn der Installationsdatenträger nicht verfügbar ist:

1. Erstellen Sie auf dem externen Computer ein Verzeichnis für die Installationsdateien.
2. Suchen Sie auf dem lokalen Computer nach folgenden Dateien:
 - rmtdbg105.exe
 - bccide.dll
 - bordbk105.dll
 - bordbk105N.dll
 - comp32x.dll
 - dbkpro100.dll
 - DCC100.DLL
 - DCC100IL.DLL
 - Borland.dbkasp.dll Diese Dateien finden Sie im Verzeichnis <C:\Programme\CodeGear\RAD Studio\5.0\Bin>.
3. Kopieren Sie die Dateien vom lokalen Computer in das Verzeichnis, das Sie auf dem externen Rechner angelegt haben.
4. Registrieren Sie auf dem externen Computer die Dateien `bordbk100.dll` und `bordbk100n.dll`, indem Sie das Registrierungs-Tool `regsvr32.exe` ausführen. Unter Windows XP geben Sie beispielsweise in der Befehlszeile `C:\Windows\System32\regsvr32.exe bordbk100.dll` und danach `C:\Windows\System32\regsvr32.exe bordbk100n.dll` ein.
5. Wenn Sie eine ASP.NET-Anwendung debuggen, kopieren Sie die Datei `Borland.dbkasp.dll` in das Verzeichnis `Install\GlobalAssemblyCache` auf dem externen Rechner. Registrieren die Datei `Borland.dbkasp.dll` im Verzeichnis `GlobalAssemblyCache`, indem Sie das Microsoft .NET-Tool `gacutil.exe` ausführen. Geben Sie beispielsweise unter Windows XP mit Microsoft .NET Framework SDK `C:\Programme\Microsoft.NET\SDK\v1.1\Bin\gacutil Borland.dbkasp.dll` ein.

Siehe auch

Überblick zum externen Debugger (siehe Seite 1441)

Verbindung für das externe Debuggen einrichten (siehe Seite 33)

Dateien für das externe Debuggen vorbereiten (siehe Seite 35)

1.24 Übersicht zum Debuggen externer Anwendungen

Mit dem externen Debugger können Sie eine RAD Studio-Anwendung debuggen, die sich auf einem externen Computer befindet. Sobald der externe Debug-Server auf dem externen Computer ausgeführt wird, können Sie über RAD Studio die Verbindung zu diesem Computer herstellen und mit dem Debuggen beginnen.

Zum Debuggen einer Anwendung auf einem externen Computer können folgende Verfahren verwendet werden:

1. Aktivieren des Debuggens auf einem Computer ohne vollständige IDE-Installation. Einzelheiten zu diesem Verfahren finden Sie im Thema Debugger auf einem externen Computer installieren (siehe Seite 30)
2. Einrichten einer Verbindung zwischen dem lokalen Computer und dem externen Rechner. Einzelheiten zu diesem Verfahren finden Sie im Thema , Verbindung für das externe Debuggen einrichten (siehe Seite 33)
3. Erzeugen von Programmdateien, die auf den externen Rechner kopiert werden. Einzelheiten zu diesem Verfahren finden Sie im Thema , Dateien für das externe Debuggen vorbereiten (siehe Seite 35)

1.25 Verbindung für das externe Debuggen einrichten

Damit das externe Debuggen möglich ist, müssen Sie zwischen dem lokalen und dem externen Computer eine TCP/IP-Verbindung einrichten. Diese Verbindung verwendet mehrere Ports, die von Windows dynamisch ausgewählt werden. Der externe Debug-Server verwendet einen Port zum Empfangen. Außerdem wird für jede Anwendung, die im Debug-Modus ausgeführt wird, ein separater Port geöffnet. Hinter einer Firewall, die Verbindungen zum passiven Port zulässt, kann der externe Debugger nicht arbeiten.

Anmerkung: Wenn auf dem externen Rechner die im Service Pack 2 von Windows XP vorhandene Firewall verwendet wird, werden Sie gefragt, ob der CodeGear-Dienst für das externe Debuggen zugelassen werden soll. Beantworten Sie diese Frage mit Ja.

Warnung: Die Verbindung zwischen RAD Studio und dem externen Debug-Server wird über einen einfachen TCP/IP-Socket hergestellt, der weder Verschlüsselung noch Authentifizierung unterstützt. Aus diesem Grund sollte der externe Debug-Server nicht auf einem Computer ausgeführt werden, auf den nicht-autorisierte Clients über ein Netzwerk zugreifen können.

So stellen Sie eine Verbindung zwischen dem lokalen Computer und dem externen Rechner her:

1. Stellen Sie sicher, dass der externe Debugger auf dem externen Rechner installiert ist.
2. Überzeugen Sie sich, dass die Programm- und Symboldateien (.tds, .rsm und .pdb) auf den externen Rechner kopiert wurden.
3. Starten Sie auf dem externen Rechner die Anwendung `rmtdbg105.exe` mit dem Argument `-listen`. `rmtdbg105.exe -listen` Der Empfänger des externen Debuggers wird gestartet und veranlasst, auf eine Verbindung mit der IDE des Host-Rechners zu warten.
4. Wählen Sie auf dem lokalen Computer **Start»Mit Prozess verbinden**. Das Dialogfeld **Mit Prozess verbinden** wird angezeigt.
5. Geben Sie den Host-Namen oder die TCP/IP-Adresse für den externen Rechner an, und klicken Sie danach auf **Aktualisieren**. Es wird eine Liste der Prozesse angezeigt, die auf dem externen Rechner ausgeführt werden. Auf diese Weise lässt sich die Konnektivität zwischen dem lokalen Computer und dem externen Rechner überprüfen.
6. Wählen Sie auf dem lokalen Computer **Start»Prozess laden»Extern**. Die Seite **Extern** des Dialogfelds **Prozess laden** wird geöffnet.
7. Geben Sie im Feld **Externer Pfad** den vollständigen Pfad für das Verzeichnis auf dem externen Rechner an, in das die Programm- und Symboldateien kopiert wurden. Der Name der Programmdatei muss im Pfad enthalten sein. Wenn Sie beispielsweise ein Programm namens `program1.exe` debuggen und dieses in ein Verzeichnis mit dem Namen `RemoteDebugFiles\Program1` auf den externen Rechner kopieren, geben Sie Folgendes an:
`C:\RemoteDebugFiles\Program1\program1.exe`
8. Geben Sie in das Feld **Externer Host** den Host-Namen oder die TCP/IP-Adresse des externen Computers ein.
9. Klicken Sie auf die Schaltfläche **Laden**. Nun wird zwischen der IDE auf dem lokalen Computer und dem Debugger auf dem externen Rechner eine Verbindung eingerichtet.

Wenn die Verbindung hergestellt ist, können Sie über die IDE des lokalen Computers die Anwendung auf dem externen Rechner debuggen, sobald diese ausgeführt wird.

Anmerkung: Sie haben keine Möglichkeit, direkt mit der externen Anwendung über den externen Debugger zu kommunizieren. Für ein interaktives Debuggen müssen Sie eine Verbindung zum externen Desktop erstellen.

Siehe auch

Überblick zum externen Debugger ([siehe Seite 1441](#))

Debugger auf einem externen Computer installieren ([siehe Seite 30](#))

Dateien für das externe Debuggen vorbereiten ([siehe Seite 35](#))

Debugger ([siehe Seite 510](#))

Linker ([siehe Seite 515](#))

1.26 Dateien für das externe Debuggen vorbereiten

Programm- und Symboldateien müssen nach ihrer Compilierung auf den externen Computer kopiert werden. Zur Generierung dieser Dateien müssen auf dem lokalen Computer die entsprechenden Dateien festgelegt werden.

So bereiten Sie Dateien für das Debuggen auf einem externen Computer vor:

1. Öffnen Sie das Projekt auf dem lokalen Computer.
2. Wählen Sie für Delphi **Projekt>Optionen>Linker**, und vergewissern Sie sich, dass die Option **Mit ext. Debug-Symbolen** aktiviert ist. Diese Festlegung veranlasst den Compiler zur Erstellung einer Symboldatei. Die folgenden Namenserweiterungen werden in Symboldateien (für Delphi-Projekte) verwendet:

Sprache	Erweiterung der Debug-Symboldateien
Delphi für Win32	.rsm
Delphi für .NET	.rsm und .pdb
C++	.tds
C#	.pdb

3. Compilieren Sie das Projekt auf dem lokalen Computer.
4. Kopieren Sie die Programm- und Symboldateien für das Projekt auf den externen Rechner.
5. Wählen Sie **Start>Prozess laden**.
6. Geben Sie im Feld **Suchpfad für Debug-Symbole** das Verzeichnis an, in das die Symboldateien kopiert werden sollen.
7. Klicken Sie auf **OK**.

Siehe auch

Überblick zum externen Debugger (siehe Seite 1441)

Debugger auf einem externen Computer installieren (siehe Seite 30)

Verbindung für das externe Debuggen einrichten (siehe Seite 33)

Die Suchreihenfolge für Debug-Symboltabellen festlegen (siehe Seite 36)

Debugger-Optionen (siehe Seite 510)

Linker (siehe Seite 515)

Symboltabellen (siehe Seite 522)

1.27 Die Suchreihenfolge für Debug-Symboltabellen festlegen

Symboltabellen werden während des Debug-Prozesses intern verwendet. RAD Studio sucht und verwendet standardmäßig alle verfügbaren Symboltabellen. Sie können jedoch die Reihenfolge bestimmen, in der diese Tabellen gesucht werden. Um das Debuggen zu beschleunigen, kann die Suche auf bestimmte Symboltabellen beschränkt werden.

Die Namenserweiterungen für Symboltabellen variieren je nach Personality.

- Delphi Win32 verwendet keine externen Symboldateien, da der Compiler die Symboltabellen im Speicher verwaltet. Wenn Sie jedoch eine externe Anwendung debuggen, müssen Sie Symboldateien mit der Erweiterung .RSM erzeugen.
- Symboldateien von Delphi.NET, VB.NET und C# haben die Namenserweiterung .PDB.
- Symboldateien von C++ besitzen die Namenserweiterung .TDS. Wenn die PE-Datei Debug-Informationen enthält, werden die externen Symboltabellen nicht verwendet.

So legen Sie die Reihenfolge fest, in der Symboltabellen gesucht werden sollen:

1. Geben Sie den allgemeinen Suchpfad für das Projekt an.
2. Geben Sie den globalen Pfad für alle Projekte an.
3. Legen Sie den sprachspezifischen Pfad für das Projekt fest.
4. Geben Sie den sprachspezifischen globalen Pfad an.

So geben Sie den allgemeinen Suchpfad für das Projekt an:

1. Wählen Sie **Projekt**▸**Optionen**▸**Debugger**▸**Symboltabellen**.
2. Geben Sie in das Feld Suchpfad für Debug-Symbole den Pfad zu der Symboltabelle ein, die der Debugger verwenden soll, oder navigieren Sie zu der gewünschten Tabelle.

Anmerkung: Wenn Sie die Suche auf bestimmte Symboltabellen beschränken möchten, fahren Sie mit dem nächsten Schritt fort. Soll der Debugger alle Pfade durchsuchen, klicken Sie auf OK. Die Festlegung des allgemeinen Projektsuchpfads ist damit abgeschlossen.

3. Deaktivieren Sie das Kontrollkästchen für das Laden aller Symbole.
4. Klicken Sie auf Neu. Das Dialogfeld Suchpfad für Symboltabelle hinzufügen wird angezeigt.
5. Geben Sie den Namen des Moduls, das Sie debuggen möchten, sowie einen oder mehrere Pfade ein, in denen die Symboltabelle für dieses Modul enthalten ist. Wenn Sie mehrere Pfade angeben, benutzen Sie ein Semikolon zur Trennung der Angaben.
6. Klicken Sie auf OK. Das Dialogfeld Suchpfad für Symboltabelle hinzufügen wird geschlossen, und das Modul und der Pfad werden in der Tabelle angezeigt.

Anmerkung: Sie können anhand dieser Liste Module und Pfade angeben, die der Debugger bei der Suche ignorieren soll. Dazu verwenden Sie einen leeren Pfad und aktivieren das Kontrollkästchen für das Laden von Symbolen für nicht spezifizierte Module.

7. Klicken Sie auf OK.

So geben Sie den globalen Pfad für alle Projekte an (nur Delphi und C++):

1. Wählen Sie **Tools**▸**Optionen**▸**Debugger-Optionen**▸**CodeGear Debugger**.

2. Geben Sie in das Feld Suchpfad für Debug-Symbole den Pfad zu der Symboltabelle ein, die der Debugger verwenden soll, oder navigieren Sie zu der gewünschten Tabelle.
3. Klicken Sie auf OK.

So geben Sie den sprachspezifischen Pfad für das Projekt an:

1. Wählen Sie **Projekt**▶**Optionen**▶**Verzeichnisse/Bedingungen**. Die Seite Verzeichnisse/Bedingungen enthält vier Felder, in denen ein Pfad für Win32- und .NET-Symboltabellen angegeben werden kann. Die Verzeichnisse werden beim Debuggen in der folgenden Reihenfolge durchsucht:
 1. Suchpfad
 2. Package-Ausgabeverzeichnis
 3. DCP/DCPIL-Ausgabeverzeichnis
 4. Ausgabeverzeichnis
2. Geben Sie in jedes dieser Felder den Pfad zu der Symboltabelle ein, die der Debugger verwenden soll, oder navigieren Sie zu der gewünschten Tabelle.
3. Klicken Sie auf OK.

So legen Sie globale Pfade fest:

1. Wählen Sie **Tools**▶**Optionen**▶**Delphi-Optionen**▶**Bibliothek (Win32 oder NET)**. Je nach Sprache enthält die Seite Bibliothek zwei oder drei Felder, in denen ein Pfad für Win32- und .NET-Symboltabellen angegeben werden kann. Die Verzeichnisse werden beim Debuggen in der folgenden Reihenfolge durchsucht:
 1. Suchpfad
 2. DCP-Ausgabeverzeichnis (in C++ nicht verwendet)
 3. Package-Ausgabeverzeichnis
2. Geben Sie in jedes dieser Felder den Pfad zu der Symboltabelle ein, die der Debugger verwenden soll, oder navigieren Sie zu der gewünschten Tabelle.
3. Klicken Sie auf OK.

Siehe auch

- Überblick zum Debuggen ([siehe Seite 1439](#))
- Überblick zum externen Debugger ([siehe Seite 1441](#))
- Symboltabellen ([siehe Seite 522](#))
- Suchpfad für Symboltabelle hinzufügen ([siehe Seite 488](#))
- CodeGear-Debugger ([siehe Seite 726](#))
- Verzeichnisse/Bedingungen ([siehe Seite 512](#))
- Bibliothek ([siehe Seite 737](#))

1.28 Verwenden der CPU-Ansicht

Die CPU-Ansicht zeigt den Quelltext Ihres Programms in der Assemblierungssprache an.

So verwenden Sie die CPU-Ansicht:

1. Führen Sie Ihr Programm aus.
2. Wählen Sie aus dem Menü **Start** ▶ **Programmausführung unterbrechen**. Die CPU-Ansicht wird angezeigt. Beachten Sie, dass bis zu fünf einzelne Fensterbereiche angezeigt werden können. Klicken Sie auf den Link *CPU-Fenster* am Ende dieses Themas, um Informationen über diese Bereiche zu erhalten.

Siehe auch

Anwendungen debuggen (siehe Seite 1439)

CPU-Fenster (siehe Seite 786)

1.29 Erweiterte Informationen zu überwachten Ausdrücken anzeigen

Beim Debuggen einer Anwendung können Sie die Werte von Elementen eines überwachten Objekts anzeigen, wenn es sich um ein komplexes Datenobjekt (wie eine Klasse, einen Record oder ein Array) handelt. Die Werte werden im Fenster Liste überwachter Ausdrücke angezeigt, wenn Sie ein überwachtes Objekt erweitern. Außerdem haben Sie die Möglichkeit, die Elemente innerhalb eines Objekts zu erweitern und deren Unterelemente und Werte anzuzeigen. Sie können alle Objektebenen erweitern. Member werden nach Vorfahr gruppiert.

So zeigen Sie erweiterte Informationen im Fenster `color="ide">Liste überwachter Ausdrücke` an:

1. Setzen Sie einen Haltepunkt auf eine gültige Quelltextzeile im Projekt. In der Leiste neben der ausgewählten Zeile wird ein Haltepunktsymbol angezeigt.
2. Wählen Sie **Start > Ausdruck hinzufügen**, um einem Objekt in der Anwendung einen Ausdruck hinzuzufügen. Der überwachte Ausdruck wird in der Liste überwachter Ausdrücke angezeigt.
3. Wählen Sie **Start > Ausführen**, um das Programm zu starten. Verwenden Sie ggf. die Funktion des Programms, die die Ausführung des Programms bis zum gesetzten Haltepunkt veranlasst. Die IDE wechselt automatisch in das Debug-Layout, und das Programm stoppt am festgelegten Haltepunkt.
4. Klicken Sie auf das Pluszeichen (+) neben dem Namen des Objekts, das Sie in die Liste der überwachten Ausdrücke eingefügt haben. Die Namen und Werte der Elemente des überwachten Objekts werden im Fenster Liste überwachter Ausdrücke angezeigt.

Siehe auch

Anwendungen debuggen (siehe Seite 1439)

Ausdruck hinzufügen (siehe Seite 17)

Quelltexthaltepunkte setzen und bearbeiten (siehe Seite 19)

Werte von Datenelementen untersuchen und ändern (siehe Seite 24)

1.30 Deployment von ASP.NET-Anwendungen

Der Deploymentmanager kann mit ASP.NET-Anwendungen für das Zusammenstellen aller .aspx-, asax-, Web.config- und anderer Assemblierungsdateien sowie von zugehörigen Assemblierungen verwendet werden.

So führen Sie das Deployment einer ASP.NET-Anwendung durch:

1. Öffnen Sie das ASP.NET-Projekt, das Sie weitergeben möchten. Klicken Sie im Fenster Projektverwaltung unter der Bibliothek, die Ihrem Projektnamen entspricht, mit der rechten Maustaste auf die Option Deployment.
2. Wählen Sie Neues ASP.NET-Deployment. Das Fenster Deploymentmanager wird geöffnet. Es gibt drei Hauptdateikategorien: ASP.NET-Auszeichnungsdateien, ausführbare Dateien und Konfigurationsdateien.
3. Die Assemblierungen für starke Namen Borland.Data.Common.dll und Borland.Data.Provider.dll sind traditionell im GAC (Global Assembly Cache) vorhanden. Fügen Sie diese beiden Dateien dem Projektverzeichnis hinzu, indem Sie unter der Bibliothek, die Ihrem Projektnamen entspricht, mit der rechten Maustaste auf Referenzen klicken und dann Referenz hinzufügen auswählen. Klicken Sie auf die beiden Dateinamen und dann auf Referenz hinzufügen. Wenn die beiden Referenzen im unteren Abschnitt des Fensters Referenz hinzufügen angezeigt werden, klicken Sie auf OK.
4. Öffnen Sie den Knoten Referenzen in der Bibliothekshierarchie. Die neuen .dll-Dateien sind aufgelistet. Wählen Sie die .dll-Dateien aus, die Sie im vorherigen Schritt hinzugefügt haben, und setzen Sie in der Projektverwaltung die Option Lokale Assemblierungseigenschaft kopieren auf True.
5. Klicken Sie Referenzen erneut mit der rechten Maustaste an, um der Assemblierung Ihre datenbankspezifische(n) .dll-Datei(en) hinzuzufügen. Wenn Sie beispielsweise MySQL Server ausführen, müssen Sie den Treiber Borland.Data.MySQL hinzufügen und in der Projektverwaltung die Option Lokale Assemblierungseigenschaft kopieren auf True setzen.
6. Compilieren Sie Ihr Projekt erneut.
7. Markieren Sie in der Projektverwaltung unter der Option Deployment Ihr Deployment-Fenster wieder.
8. Geben Sie den Pfad zu dem Zielrechner ein, auf dem Ihre Anwendung weitergegeben werden soll. Klicken Sie dann auf das Symbol neben jedem Dateinamen. Die Dateien werden dadurch auf die Seite Ziel des Fensters verlagert. Wenn all Ihre Quelltextdateien auf der Seite Ziel des Fensters angezeigt werden, wurden Sie an den Zielrechner weitergegeben.

Siehe auch

Deployment von Anwendungen (siehe Seite 1443)

Deploymentmanager verwenden

1.31 Deployment von Anwendungen

Dieser Abschnitt enthält Anleitungen zum Deployment von Anwendungen. Das Deployment von Anwendungen kann manuell oder mit Hilfe des Deploymentmanagers erfolgen. Gegenwärtig kann der Deploymentmanager nur bei ASP.NET-Anwendungen eingesetzt werden.

1.32 Code-Folding verwenden

Mit Hilfe des Code-Folding können Sie bestimmte Bereiche des Quelltextes ein- und ausblenden, der sich damit einfacher durchsuchen und lesen lässt. RAD Studio generiert Quelltext, der bereits Code-Folding-Bereiche enthält. Sie können aber auch eigene Bereiche nach Bedarf hinzufügen.

So blenden Sie Quelltext ein und aus:

1. Klicken Sie im Quelltext-Editor auf das Minuszeichen (-) links neben einem Quelltextblock, um diesen Quelltext auszublenden.
2. Klicken Sie auf das Pluszeichen (+), um den Quelltext einzublenden.

Tip: Um das Code-Folding während der aktuellen Arbeitssitzung zu deaktivieren, halten Sie **STRG+UMSCHALT** gedrückt und drücken dann **K** und **O**. Um den nächsten Block auszublenden, halten Sie **STRG+UMSCHALT** gedrückt und drücken dann **K** und **E**. Um den nächsten Block einzublenden, halten Sie **STRG+UMSCHALT** gedrückt und drücken dann **K** und **U**. Um den gesamten Quelltext einzublenden, halten Sie **STRG+UMSCHALT** gedrückt und drücken dann **K** und **A**.

So fügen Sie einen Code-Folding-Bereich hinzu:

1. Schließen Sie im Quelltext-Editor einen Quelltextblock in die folgenden Präprozessordirektiven ein:

```
{$region 'Optionaler Text, der angezeigt wird, wenn der Codeblock ausgeblendet ist'}
.
.
{$endregion}
#region Optionaler Text, der angezeigt wird, wenn der Codeblock ausgeblendet ist
.
.
#endregion
#pragma region Optionaler Text
.
.
#pragma end_region
```

Der Bereich wird mit einem Minuszeichen (-) markiert.

2. Klicken Sie auf das Minuszeichen (-), um den Bereich auszublenden.

Siehe auch

Quelltext-Editor anpassen (↗ siehe Seite 44)

Code Insight verwenden (↗ siehe Seite 48)

1.33 Quelltext-Templates erstellen

Im Quelltext-Editor können Sie bevorzugte Quelltext-Konstrukte in den Template-Manager einfügen, um eine Bibliothek häufig verwendeter Templates zu erstellen.

So fügen Sie über Menübefehle eine Quelltext-Template hinzu:

1. Wählen Sie im Quelltext-Editor den Befehl **Datei**▶**Neu**▶**Weitere**▶**Andere Dateien**, und klicken Sie auf das Symbol Quelltext-Template.
2. Geben Sie den Template-Namen, eine Beschreibung, den Autor und Sprachattribute ein. Geben Sie dann den Quelltext für die Template zwischen den Tags `<![CDATA[]]>` und `</code>` ein.

Anmerkung: Die Felder **Name** und **Language** in der Template sind erforderliche Felder.

3. Wählen Sie den Befehl Speichern im Menü **Datei** des Quelltext-Editors (oder drücken Sie STRG+S). Die neue Template wird in der IDE-Hierarchie im Template-Manager angezeigt. Sie wird standardmäßig im Verzeichnis `\5.0\Objrepos\code_templates\` gespeichert.

So fügen Sie eine Quelltext-Template über den Template-Manager hinzu:

1. Wählen Sie im Quelltext-Editor den Befehl **Ansicht**▶**Templates**.
2. Klicken Sie im Fenster Template-Manager auf die Schaltfläche **Neu**. Daraufhin wird im Hauptfenster des Quelltext-Editors eine XML-Gliederung für eine Quelltext-Template angezeigt. Sie können auch Quelltext im Editor auswählen, bevor Sie auf die Schaltfläche **Neu** klicken.
3. Geben Sie den Template-Namen, eine Beschreibung, den Autor und Sprachattribute ein. Geben Sie dann den Quelltext für die Template zwischen den Tags `<![CDATA[]]>` und `</code>` ein.

Anmerkung: Die Felder **Name** und **Language** in der Template sind erforderliche Felder.

4. Wählen Sie den Befehl Speichern im Menü **Datei** des Quelltext-Editors (oder drücken Sie STRG+S). Die neue Template wird in der IDE-Hierarchie im Template-Manager angezeigt. Sie wird standardmäßig im Verzeichnis `\5.0\Objrepos\code_templates\` gespeichert.

Siehe auch

Quelltext-Templates verwenden (↗ siehe Seite 50)

Quelltext-Editor anpassen (↗ siehe Seite 44)

1.34 Quelltext-Editor anpassen

CodeGear RAD Studio bietet die Möglichkeit den Quelltext-Editor anzupassen. So lassen sich zum Beispiel die Tastaturbelegung, die Schriftarten, Randbreiten, Farben, die Syntaxhervorhebung und die Einzugsarten ändern.

So ändern Sie die allgemeinen Optionen des Quelltext-Editors:

1. Wählen Sie **Tools > Optionen**.
2. Klicken Sie auf Editor-Optionen.
3. Nehmen Sie die gewünschten Änderungen vor.
4. Klicken Sie auf OK, damit die Änderungen im Quelltext-Editor wirksam werden.

Siehe auch

Code-Folding verwenden (↗ siehe Seite 42)

Code Insight verwenden (↗ siehe Seite 48)

1.35 Tastatur-Makro aufzeichnen

Sie können eine Abfolge von Tastenanschlägen als Makro aufzeichnen, während Sie Quelltext bearbeiten. Nach der Aufzeichnung des Makros können die aufgezeichneten Tastenanschläge während der aktuellen Arbeitssitzung in der IDE beliebig oft wiederholt werden.

So zeichnen Sie ein Makro auf:

1. Klicken Sie im Quelltext-Editor unten im Quelltextfenster auf die Schaltfläche Makro aufzeichnen , um den Aufzeichnungsmodus zu aktivieren.
2. Drücken Sie die Tasten, die aufgezeichnet werden sollen.
3. Ist die Eingabe der Tastenfolge beendet, klicken Sie auf die Schaltfläche Aufzeichnung beenden .
4. Um ein weiteres Makro aufzuzeichnen, wiederholen Sie die vorherigen Schritte.

Anmerkung: Das neu aufgezeichnete Makro ersetzt das zuvor aufgezeichnete Makro.

Das Makro steht nun in der aktuellen Arbeitssitzung in der IDE zur Verfügung.

So führen Sie ein Makro aus:

1. Setzen Sie den Cursor im Quelltext-Editor an diejenige Position im Quelltext, an der das Makro ausgeführt werden soll.
2. Klicken Sie auf die Schaltfläche Makro abspielen , um das Makro auszuführen. Wenn die Schaltfläche nur schemenhaft sichtbar ist, steht kein Makro zur Verfügung.

1.36 Positionsmarken verwenden

Sie können eine Stelle im Quelltext mit einer Positionsmarke markieren und diese von einer beliebigen Stelle in der Datei aus an springen. Es können bis zu zehn Positionsmarken definiert werden. Die Positionsmarken werden zusammen mit der Datei gespeichert und stehen nach dem erneuten Öffnen wieder im Quelltext-Editor zur Verfügung.

So setzen Sie eine Positionsmarke:

1. Klicken Sie im Quelltext-Editor mit der rechten Maustaste auf die Quelltextzeile, in der eine Positionsmarke gesetzt werden soll. Das Kontextmenü des Quelltext-Editors wird geöffnet.
2. Wählen Sie **Positionsmarken umschalten**▶**Positionsmarke n**, wobei *n* eine Zahl zwischen 0 und 9 ist. Ein Positionsmarkensymbol  wird in der Leiste am linken Rand des Quelltext-Editors angezeigt.

Tip: Sie können eine Positionsmarke auch mit Hilfe von Tastaturkürzeln definieren. Drücken Sie dazu die Tastenkombination **STRG+UMSCHALTTASTE** und eine Zahl zwischen 0 und 9.

So springen Sie zu einer Positionsmarke:

1. Klicken Sie im Quelltext-Editor mit der rechten Maustaste, um das Kontextmenü zu öffnen.
2. Wählen Sie **Zu Positionsmarke gehen**▶**Positionsmarke n**, wobei *n* eine Zahl zwischen 0 und 9 ist.

Tip: Sie können eine Positionsmarke auch mit Hilfe von Tastaturkürzeln an springen. Drücken Sie dazu **STRG** und die Nummer der gewünschten Positionsmarke. Mit **STRG+1** gelangen Sie beispielsweise zu der Quelltextzeile, in der die Positionsmarke 1 platziert wurde.

So entfernen Sie eine Positionsmarke:

1. Klicken Sie im Quelltext-Editor mit der rechten Maustaste, um das Kontextmenü zu öffnen.
2. Wählen Sie **Positionsmarken umschalten**▶**Positionsmarke n**, wobei *n* die Nummer der Positionsmarke ist, die entfernt werden soll. Die Positionsmarke wird nun aus der Leiste am linken Rand des Quelltext-Editors entfernt.

Tip: Wenn Sie alle Positionsmarken in einer Datei löschen möchten, wählen Sie **Positionsmarken löschen**.

1.37 Klassen vervollständigung verwenden

Die Klassen vervollständigung dient zur Automatisierung der Definition von neuen Klassen, wobei für die zu deklarierenden Delphi-Klassenelemente Skeleton-Quelltext generiert wird.

So nutzen Sie die Klassen vervollständigung:

1. Deklarieren Sie im Quelltext-Editor im **interface**-Abschnitt einer Unit eine Klasse. Geben Sie beispielsweise Folgendes ein:

```
type TMyButton = class(TButton)
  property Size: Integer;
  procedure DoSomething;
end;
```

2. Klicken Sie mit der rechten Maustaste auf die Klassendeklaration, und wählen Sie Klasse beim Cursor vervollständigen.

Tip: Sie können die Klassen vervollständigung auch aktivieren, indem Sie den Cursor in die Klassendeklaration stellen und die Tastenkombination **STRG+UMSCHALT+C** drücken.

Bei der Klassen vervollständigung werden automatisch die Bezeichner **read** und **write** zu den Deklarationen aller Eigenschaften hinzugefügt, welche diese benötigen. Ferner wird ein Quelltext-Grundgerüst für alle Methoden der Klasse zum **implementation**-Abschnitt hinzugefügt.

Tip: Sie können mit dieser Funktion auch **interface**-Deklarationen für Methoden erzeugen lassen, die im Abschnitt **implementation** definiert sind.

Nach der Aktivierung der Klassen vervollständigung sieht der obige Beispielquelltext folgendermaßen aus:

```
type TMyButton = class(TButton)
  private
    FSize: Integer;
  procedure SetSize(const Value: Integer);
  published
    property Size: Integer read FSize write set_Size;
    procedure DoSomething;
end;
```

Im Abschnitt **implementation** wird das folgende Quelltext-Grundgerüst hinzugefügt:

```
{ TMyButton }

procedure TMyButton.DoSomething;
begin
end;

procedure TMyButton.SetSize(const Value: Integer);
begin
  FSize := Value;
end;
```

Wenn Deklarationen und Implementierungen alphabetisch sortiert werden, wird diese Reihenfolge durch Verwendung der Programmierhilfunktion nicht geändert. Ansonsten werden neue Routinen an das Ende des Abschnitts **implementation** der Unit gestellt, neue Deklarationen in als **private** deklarierte Abschnitte am Anfang der Klassendeklaration eingefügt.

Tip: Die Option Unvollständige Eigenschaften vervollständigen auf der Seite **Tools>Optionen>Explorer** bestimmt, ob die Klassen vervollständigung auch Eigenschaftsdeklarationen vervollständigt.

1.38 Code Insight verwenden

Als Code Insight (oder Programmierhilfe) wird ein Satz von Funktionen für den Quelltext-Editor und HTML-Tag-Editor bezeichnet, der Funktionen zur Code-Vervollständigung, zur Anzeige von Parameterlisten im Quelltext sowie Kurzhinweise für Ausdrücke und Symbole umfasst.

Im Hinweisfeld werden keine Deklarationen von Interface-Methoden angezeigt, auf die in Lese- und Schreibroutinen für Eigenschaften Bezug genommen wird. Die Liste enthält nur Eigenschaften und eigenständige Methoden, die im Interface-Typ deklariert sind.

So aktivieren Sie Code Insight (allgemeine Anleitung):

1. Wählen Sie **Tools**▶**Optionen**▶**Code Insight**.
2. Legen Sie auf der Seite Code Insight die gewünschten Optionen und Farbvorgaben fest. Die folgenden Anleitungen erläutern einige Code Insight Einstellungen detailliert.
3. Klicken Sie auf OK.

So aktivieren Sie die Programmierhilfe:

1. Wählen Sie **Tools**▶**Optionen**▶**Code Insight**.
2. Markieren Sie auf der Seite Code Insight das Kontrollkästchen Programmierhilfe.
3. Um im Quelltext-Editor eine Liste mit Typen, Eigenschaften, Methoden und Ereignissen anzuzeigen, geben Sie nach dem Namen eines Objekts oder einer Klasse entweder einen Punkt (.) (für Delphi und C++) oder einen Pfeil (→) (für C++) ein. Zum Anzeigen der Eigenschaften, Methoden und Ereignisse, die in einer Klasse verfügbar sind, geben Sie den Namen einer Variablen ein und drücken **STRG+LEER**.
4. Wählen Sie das angezeigte Element aus, das die Klasse oder das Objekt vervollständigen soll, und drücken Sie die Taste **EINGABE**. Um die Programmierhilfe abzubrechen, drücken Sie entweder die Taste **RÜCK** oder **ESC**.

Beispiele für Code Insight

1. Bei Verwendung von C++ geben Sie den Namen einer Variable ein, die einen Zeiger auf eine Klasseninstanz repräsentiert, und drücken **STRG+LEER**. Daraufhin werden die Eigenschaften, Methoden und Ereignisse für die Klasse angezeigt. Zum Aufrufen der Programmierhilfe für einen Zeigertyp muss der Zeiger zuvor dereferenziert werden. Geben Sie beispielsweise `this` für C++ oder `self` für Delphi ein.
2. In C++ geben Sie einen Pfeil (→) für einen Zeiger auf ein Objekt ein. Für Typen, die keine Zeiger sind, können Sie auch den Namen und einen Punkt eingeben, um die Liste abgeleiteter und virtueller Eigenschaften, Methoden und Ereignisse zu erhalten. Für Delphi geben Sie z.B. Folgendes ein: `var test: TRect; : : begintest`. Für C++ geben Sie `TRect test; test` ein.
3. Geben Sie einen Zuweisungsoperator oder den Anfang einer Zuweisungsanweisung ein, und drücken Sie **STRG+LEER**, um eine Liste der verfügbaren Werte für die Variable anzuzeigen.
4. Geben Sie einen Prozedur-, Funktions- oder Methodenaufruf ein, und drücken **STRG+LEER**, um die Methode und die Liste der Argumente anzuzeigen.
5. Geben Sie eine Record-Deklaration ein, um eine Liste mit Feldern anzuzeigen. (Dies entspricht Schritt 1, nur dass hier Records und nicht Klassen verwendet werden).

So aktivieren Sie Code-Parameter:

1. Wählen Sie **Tools**▶**Optionen**▶**Code Insight**.
2. Markieren Sie das Kontrollkästchen Code-Parameter.
3. Damit die Programmierhilfe im Quelltext-Editor Methodenargumente anzeigt, geben Sie einen Methodennamen und eine

geöffnete Klammer (()) ein.

So aktivieren und nutzen Sie die Auswertung durch Kurzhinweis:

1. Wählen Sie **Tools**▶**Optionen**▶**Code Insight**.
2. Markieren Sie das Kontrollkästchen Auswertung durch Kurzhinweis.
3. Um bei einer Unterbrechung des Debuggens den aktuellen Wert einer Variable anzuzeigen, zeigen Sie Quelltext-Editor mit dem Mauszeiger auf einen Variablennamen.

So aktivieren und nutzen Sie die Symbolinformation durch Kurzhinweis:

1. Wählen Sie **Tools**▶**Optionen**▶**Code Insight**.
2. Markieren Sie das Kontrollkästchen Symbolinformation durch Kurzhinweis.
3. Um während der Bearbeitung des Quelltextes die Deklaration eines Bezeichners zu sehen, zeigen Sie im Quelltext-Editor darauf.

Siehe auch

Quelltext-Templates verwenden (↗ siehe Seite 50)

Code-Folding verwenden (↗ siehe Seite 42)

Quelltext-Editor anpassen (↗ siehe Seite 44)

Quelltext-Templates verwenden (↗ siehe Seite 50)

Mit dem HTML-Tag-Editor arbeiten

usingcodeinsight.xml

1.39 Quelltext-Templates verwenden

Quelltext-Templates sind wieder verwendbare Quelltextanweisungen, auf die über den Quelltext-Editor zugegriffen werden kann. Sie können vordefinierte Quelltextsegmente in Ihren eigenen Quelltext einfügen oder eigene Code-Snippets dem Fenster Template hinzufügen.

Anmerkung: Wenn eine Template mehrere editierbare Sprungpunkte enthält, erfolgt automatisch ein Wechsel in den **Sync-Bearbeitungsmodus**, sobald Sie die Template in den Quelltext einfügen. Mit Hilfe der Sprungpunkte können Sie mit den Tasten TAB und UMSCHALT+TAB zwischen verschiedenen Bereichen der Template hin- und herwechseln. Wenn Sie auf dem letzten Sprungpunkt ESC, EINGABE (oder die TAB-Taste) drücken, wird der **Sync-Bearbeitungsmodus** beendet und der Quelltext-Editor wieder in den normalen Bearbeitungsmodus versetzt. Weitere Informationen zum **Sync-Bearbeitungsmodus** finden Sie über den Link am Ende dieses Themas.

So fügen Sie eine vorhandene Quelltext-Template in den Quelltext ein:

1. Wählen Sie im Quelltext-Editor den Befehl **Ansicht▶Templates**.
2. Erweitern Sie den Baum im Template-Manager für die verwendete Sprache, indem Sie auf das Pluszeichen vor dem Namen der Sprache klicken.
3. Setzen Sie den Cursor an die Stelle im Quelltext, an der die Template eingefügt werden soll.
4. Wählen Sie die gewünschte Template im Fenster Template-Manager aus.
5. Klicken Sie im Fenster Template-Manager auf die Schaltfläche Ausführen.

Nach dem Einfügen der Template müssen Sie Daten, Variablen, Methoden oder andere für den Quelltext spezifische Informationen eingeben. Für einige Templates können Sie die Funktion Code-Vervollständigung verwenden, wie im Folgenden beschrieben.

So nutzen Sie die Code-Vervollständigung für die Template:

1. Drücken Sie an einem Sprungpunkt in der Template STRG+LEER, um das Fenster Code-Vervollständigung zu öffnen.
- Sie können den Quelltext auf zwei Arten in eine Template einbetten. Verwenden Sie das Verfahren, das Ihrem Arbeitsstil entgegenkommt.

So betten Sie Quelltext mit Hilfe der Maus in eine Template ein:

1. Wählen Sie im Quelltext-Editor den Quelltext aus, der in die Template eingebettet werden soll.
2. Klicken Sie mit der rechten Maustaste, und wählen Sie Umgeben. Eine Liste der geeigneten Templates wird angezeigt.
3. Wählen Sie eine Template in der Liste aus.

So betten Sie Quelltext mit Hilfe des Template-Managers in eine Template ein:

1. Wählen Sie im Quelltext-Editor den Befehl **Ansicht▶Templates**.
2. Erweitern Sie den Baum im Template-Manager für die verwendete Sprache, indem Sie auf das Pluszeichen vor dem Namen der Sprache klicken.
3. Wählen Sie die gewünschte Template im Fenster Template-Manager aus.
4. Wählen Sie im Quelltext-Editor den Quelltext aus, der in die Template eingebettet werden soll.
5. Klicken Sie im Fenster Template-Manager auf die Schaltfläche Ausführen.

Siehe auch

- Quelltext-Templates erstellen ( siehe Seite 43)
- Sync-Bearbeitungsmodus verwenden ( siehe Seite 54)
- Code-Folding verwenden ( siehe Seite 42)
- Quelltext-Editor anpassen ( siehe Seite 44)
- Programmierhilfe verwenden ( siehe Seite 48)
- Mit dem HTML-Tag-Editor arbeiten

1.40 Mit der Versionsverwaltung arbeiten

Mit der Versionsverwaltung können Sie verschiedene Versionen einer Datei anzeigen und miteinander vergleichen, einschließlich mehrerer Sicherungsversionen, gespeicherter lokaler Änderungen und Inhalt des Bearbeitungspuffers mit nicht gespeicherten Änderungen.

Zur Vereinfachung werden die Funktionen der Versionsverwaltung in den folgenden Anleitungen anhand einer kleinen Textdatei vorgestellt. Die Versionsverwaltung steht jedoch für die meisten Dateien, einschließlich Quelltext- und HTML-Dateien, zur Verfügung.

So können Sie Dateiversionen erstellen und auf der Seite `color="ide" Inhalt anzeigen`:

1. Wählen Sie **Tools**▸**Optionen**▸**Editor-Optionen**, und stellen Sie sicher, dass die Option Sicherungsdateien erstellen ausgewählt ist.
2. Wählen Sie **Datei**▸**Neu**▸**Weitere**▸**Andere Dateien**▸**Text**, und klicken Sie auf OK, um im Quelltext-Editor eine leere Textdatei anzuzeigen.
3. Geben Sie in der ersten Zeile der Datei `Erste Textzeile` ein, und speichern Sie die Datei unter einem beliebigen Namen in einem beliebigen Verzeichnis.
4. Geben Sie in der zweiten Zeile `Zweite Textzeile` ein, und speichern Sie die Datei.
5. Geben Sie in der dritten Zeile `Dritte Textzeile` ein, und speichern Sie die Datei. Das aktuelle Verzeichnis enthält nun in einem verborgenen Unterverzeichnis mit dem Namen `__history` drei verschiedene Versionen der Datei.
6. Klicken Sie auf die Registerkarte Historie, die sich neben der Registerkarte Code befindet. Die Versionsliste im oberen Teil der Registerkarte Inhalt enthält drei Versionen der Datei. Die erste Version trägt den Namen `~1~`, die zweite den Namen `~2~`. Die aktuelle Version wird unter der Bezeichnung **Datei** angezeigt. In der Quellansicht im unteren Teil der Registerkarte sehen Sie den Inhalt der ausgewählten Datei.
7. Wählen Sie die verschiedenen Versionen aus, um ihren Inhalt anzuzeigen.
8. Klicken Sie auf die Registerkarte Code, um zum Quelltext-Editor zurückzukehren. Geben Sie dann in der vierten Zeile der Datei `Vierte Textzeile` ein, aber speichern Sie die Datei **nicht**. Die Änderung wird im Editorpuffer abgelegt, aber nicht in der Datei gespeichert.
9. Lesen Sie die folgenden Beschreibungen der Schaltflächen in der Symbolleiste, und fahren Sie anschließend mit der nächsten Anleitung fort, um die erstellten Dateiversionen miteinander zu vergleichen.

Tip: Sie können die Spalten auf den Seiten der Versionsverwaltung sortieren, indem Sie auf die gewünschte Spaltenüberschrift klicken.

Nachstehend finden Sie eine kurze Beschreibung der Schaltflächen in der Symbolleiste der Versionsverwaltung. Beachten Sie, dass manche dieser Schaltflächen nur auf bestimmten Seiten der Versionsverwaltung verfügbar sind.

Tip: Die Funktionen der Symbolleistenschaltflächen können auch über Kontextmenüs auf den Seiten der Versionsverwaltung aufgerufen werden.

So vergleichen Sie Dateiversionen auf der Seite `color="ide" Unterschied`:

1. Laden Sie die Datei, die Sie in der vorherigen Anleitung erstellt haben, und klicken Sie auf die Registerkarte Historie.
2. Klicken Sie auf die Registerkarte Unterschiede am unteren Rand der Versionsverwaltung. Oben auf der Seite werden in den Fenstern Unterschiede in und Zu die Dateiversionen angezeigt, die Sie vergleichen können. Unten auf der Seite sind die gelöschten Quelltextzeilen hervorgehoben und mit einem Minuszeichen (-) gekennzeichnet. Hinzugefügte Zeilen sind mit einem Pluszeichen (+) markiert. Die Hervorhebungsfarben hängen von den Farben ab, die für den Quelltext-Editor eingestellt sind.

3. Wählen Sie die gewünschten Dateiversionen in den Fenstern Unterschiede in und Zu aus. Die Ergebnisse sind in der Quellanzeige zu sehen.

So machen Sie eine frühere Version zur aktuellen Version:

1. Laden Sie die Datei, die Sie zuvor verwendet haben, und klicken Sie auf die Registerkarte Inhalt.
2. Klicken Sie mit der rechten Maustaste auf die Version  der Datei, und wählen Sie Zurücksetzen, oder klicken Sie auf die Schaltfläche  in der Symbolleiste. In einem Bestätigungsdialogfeld werden Sie darauf hingewiesen, dass durch das Zurücksetzen alle nicht gespeicherten Änderungen im Puffer verloren gehen.
3. Klicken Sie im Bestätigungsdialogfeld auf Ja. Die Version  ist nun die aktuelle Version.
4. Kehren Sie zum Quelltext-Editor zurück, und speichern Sie die Änderung.

Tip: Der Befehl Zurücksetzen steht auch auf der Seite Info zur Verfügung.

Siehe auch

Ein erster Blick auf die IDE (siehe Seite 1363)

Versionsverwaltung

1.41 Sync-Bearbeitungsmodus verwenden

Im Sync-Bearbeitungsmodus lassen sich Bezeichner, die mehrfach im ausgewählten Quelltext vorhanden sind, gleichzeitig bearbeiten. Beispielsweise müssen Sie in einer Prozedur, in der `Label1` dreimal vorkommt, nur den ersten Bezeichner bearbeiten, die beiden anderen Vorkommen werden automatisch entsprechend geändert.

So verwenden Sie den Sync-Bearbeitungsmodus:

1. Wählen Sie im Quelltext-Editor einen Block aus, in dem ein Bezeichner mehrfach vorhanden sind.
2. Klicken Sie in der Leiste am linken Fensterrand auf das Symbol Sync-Bearbeitungsmodus . Der erste der mehrfach vorhandenen Bezeichner wird optisch hervorgehoben, die anderen werden unterstrichen. Der Cursor befindet sich auf dem ersten Bezeichner. Wenn der Quelltext mehrere Gruppen mit identischen Bezeichnern enthält, können Sie mit der TAB-Taste zwischen den Bezeichnern im ausgewählten Bereich hin- und herwechseln.
3. Beginnen Sie mit der Bearbeitung des ersten Bezeichners. Sobald Sie diesen ändern, wird dieselbe Änderung automatisch an den anderen Bezeichnern durchgeführt. Der Bezeichner wird standardmäßig ersetzt. Um den Bezeichner zu ändern, ohne ihn zu ersetzen, verwenden Sie die Pfeiltasten, bevor Sie mit der Eingabe beginnen.
4. Nach der Änderung der Bezeichner können Sie den Sync-Bearbeitungsmodus beenden, indem Sie auf das Symbol Sync-Bearbeitungsmodus klicken oder die Taste `ESC` drücken.

Anmerkung: Die Sync-Bearbeitungsfunktion ermittelt mehrfach vorhandene Bezeichner anhand eines Textstring-Vergleichs. Es erfolgt also keine Analyse der Bezeichner. So wird beispielsweise nicht zwischen zwei gleichnamigen Bezeichnern verschiedener Typen unterschieden, die sich in unterschiedlichen Gültigkeitsbereichen befinden. Aus diesem Grund eignet sich der Sync-Bearbeitungsmodus für kleine Quelltextabschnitte, etwa eine Methode oder eine Textseite. Zur Änderung größerer Quelltextabschnitte sollten Sie das Refactoring verwenden.

1.42 Komponenten in ein Formular einfügen

So fügen Sie eine Komponente in ein Formular ein:

1. Wählen Sie in der Tool-Palette eine visuelle oder nicht-visuelle Komponente aus.
2. Fügen Sie die Komponente durch Doppelklicken oder Ziehen mit der Maus in das Formular ein. Wenn Sie eine nicht-visuelle Komponente in das Formular einfügen, wird die Komponentenablage am unteren Rand des Designers angezeigt.
3. Wiederholen Sie Schritt 1 und 2, um weitere Komponenten hinzuzufügen.
4. Das gepunktete Raster im Formular unterstützt Sie bei der Ausrichtung der Komponenten.

Siehe auch

[Einführung](#) (siehe Seite 1353)

[Ein Projekt starten](#) (siehe Seite 1369)

[Ein Projekt erstellen](#) (siehe Seite 61)

[Projektoptionen einstellen](#) (siehe Seite 75)

[Eigenschaften und Ereignisse festlegen](#) (siehe Seite 77)

1.43 Referenzen hinzufügen

Sie können bereits vorhandene COM-Server und ActiveX-Elemente in verwaltete Anwendungen integrieren, indem Sie Referenzen zu unverwalteten DLLs in Ihr Projekt einfügen und dann wie bei verwalteten Assemblierungen die Typen einfach durchsuchen.

So fügen Sie Referenzen hinzu:

1. Wählen Sie aus dem Hauptmenü **Projekt>Referenz hinzufügen**. Das Dialogfeld Referenzen hinzufügen wird geöffnet.
2. Wählen Sie entweder eine vorhandene COM-Typbibliothek oder ein ActiveX-Element für die Integration in Ihre verwaltete Anwendung aus.
3. Klicken Sie auf Referenz hinzufügen. Die Referenz wird in das Textfeld eingefügt.
4. Klicken Sie auf OK.

Tip: Sie können auch in der Projektverwaltung mit der rechten Maustaste auf den Ordner Referenzen klicken und Referenz hinzufügen wählen.

1.44 Dateien hinzufügen und entfernen

Sie können verschiedene Dateitypen zu Ihren Projekten hinzufügen und daraus entfernen.

So fügen Sie eine Datei in ein Projekt ein:

1. Wählen Sie **Projekt>Dem Projekt hinzufügen**. Das Dialogfeld Dem Projekt hinzufügen wird angezeigt.
2. Wählen Sie eine Datei aus, und klicken Sie auf Öffnen. Die Datei wird unterhalb des Knotens `Project.exe` in der Projektverwaltung angezeigt.

So entfernen Sie eine Datei aus einem Projekt:

1. Wählen Sie **Projekt>Aus dem Projekt entfernen**. Das Dialogfeld Aus dem Projekt entfernen wird geöffnet.
2. Markieren Sie die Datei(en), die Sie entfernen möchten, und klicken Sie auf OK.

Siehe auch

Einführung (siehe Seite 1353)

Ein Projekt erstellen (siehe Seite 61)

1.45 Templates in die Objektablage einfügen

Sie können in die Objektablage eigene Projekte als Templates einfügen, um diese wieder zu verwenden oder für andere Entwickler bereitzustellen. Die mehrfache Verwendung von Objekten hat den Vorteil, dass Sie Programmfamilien mit gemeinsamer Benutzeroberfläche und Funktionalität erstellen können. Dadurch sparen Sie nicht nur Entwicklungszeit, sondern verbessern auch die Qualität Ihrer Produkte.

So fügen Sie der Objektablage eine Template hinzu:

1. Speichern Sie das Projekt.
2. Wählen Sie **Projekt > Der Objektablage hinzufügen**.
3. Geben Sie im Dialogfeld den Projektnamen, eine Beschreibung und die Autoreninformationen ein.
4. Klicken Sie auf Durchsuchen, um für das gespeicherte Projekt ein Symbol auszuwählen.
5. Klicken Sie auf OK.

Siehe auch

Einführung (siehe Seite 1353)

Dateien hinzufügen und entfernen (siehe Seite 57)

1.46 Referenzen in lokalen Pfad kopieren

Zur Laufzeit müssen sich Assemblierungen im Ausgabepfad des Projekts oder zum Deployment im globalen Assemblierungs-Cache (GAC) befinden. Wenn Projekte eine Referenz auf ein Objekt enthalten, das sich nicht an einer der beiden Positionen befindet, muss die Referenz in den betreffenden Ausgabepfad kopiert werden.

So kopieren Sie Referenzen in einen lokalen Pfad:

1. Klicken Sie in der Projektverwaltung mit der rechten Maustaste auf eine im Ordner Referenzen enthaltene Assemblierungs-DLL.
2. Stellen Sie die Option Lokal kopieren so ein, dass die Datei in das Ausgabeverzeichnis kopiert wird.

Anmerkung: In der IDE bleibt die Einstellung Lokal kopieren so lange aktiv, bis Sie sie ändern.

Siehe auch

Ein erster Blick auf die IDE (siehe Seite 1363)

1.47 Komponenten-Templates erzeugen

Sie können ausgewählte, vorkonfigurierte Komponenten im aktuellen Formular als wiederverwendbare Komponenten-Templates speichern, die in der Tool-Palette zur Verfügung stehen.

So erstellen Sie eine Komponenten-Template:

1. Plazieren Sie Komponenten in einem Formular.
2. Weisen Sie den Komponenten im Objektinspektor die gewünschten Eigenschaften und Ereignisse zu.
3. Markieren Sie die Komponenten, die Sie als Komponenten-Template speichern möchten. Sie können mehrere Komponenten auswählen, indem Sie den Mauszeiger über die betreffenden Komponenten ziehen.

Tip: Zur Auswahl aller Komponenten des Formulars wählen Sie **Bearbeiten**▶**Alles auswählen**.

Markierte Komponenten haben einen grauen Griff an jeder Ecke.

4. Wählen Sie **Komponenten**▶**Komponenten-Template erzeugen**. Das Dialogfeld Komponenten-Template erzeugen wird angezeigt.
5. Geben Sie einen Namen, eine Kategorie der Tool-Palette und ein Symbol für die Template an.
6. Klicken Sie auf OK.

Die neue Template wird sofort unter der von Ihnen angegebenen Kategorie in der Tool-Palette angezeigt.

So verwenden Sie eine Komponenten-Template:

1. Öffnen Sie das Formular, in das Sie Komponenten der Komponenten-Template einfügen möchten.
2. Doppelklicken Sie in der Tool-Palette auf das Symbol der Komponenten-Template. Die in der Template enthaltenen Komponenten werden zusammen mit ihren vorkonfigurierten Eigenschaften und Ereignissen in das Formular eingefügt. Nach dem Einfügen können Sie nach Bedarf die Position der Komponenten ändern, ihre Eigenschaften neu zuweisen und Ereignisbehandlungsroutine erstellen oder bearbeiten.

So löschen Sie eine Komponenten-Template:

1. Doppelklicken Sie in der Tool-Palette mit der rechten Maustaste auf die Komponenten-Template, um ein Kontextmenü zu öffnen.
2. Wählen Sie den Befehl Schaltfläche [Template-Name] löschen. Die Komponenten-Template wird sofort aus der Tool-Palette gelöscht.

1.48 Ein Projekt erstellen

So fügen Sie ein neues Projekt hinzu:

1. Wählen Sie **Projekt > Neues Projekt hinzufügen**. Das Dialogfeld Objektgalerie wird geöffnet.
2. Wählen Sie ein Projekt aus, und klicken Sie auf OK. Das Projekt wird der Projektverwaltung hinzugefügt.

So fügen Sie ein vorhandenes Projekt hinzu:

1. Wählen Sie **Projekt > Existierendes Projekt hinzufügen**. Das Dialogfeld Projekt öffnen wird geöffnet
2. Wählen Sie ein vorhandenes Projekt aus, und klicken Sie auf Öffnen.

Siehe auch

Ein Projekt starten ( siehe Seite 1369)

Dateien hinzufügen und entfernen ( siehe Seite 57)

Komponenten in ein Formular einfügen ( siehe Seite 55)

Projektoptionen einstellen ( siehe Seite 75)

Eigenschaften ( siehe Seite 77)

1.49 Formular anpassen

So passen Sie ein Formular an:

1. Wählen Sie **Tools > Optionen**.
2. Klicken Sie im Dialogfeld Optionen auf Windows Formular-Designer.
3. Aktivieren oder Deaktivieren Sie die Funktionen Am Raster ausrichten und Raster anzeigen mit Hilfe der zugehörigen Kontrollkästchen.
4. Wählen Sie einen Stil für die geschweiften Klammern aus.
5. Klicken Sie auf OK.

Tip: Änderungen wirken sich nur auf diejenigen Formulare aus, die nach der Änderung dieser Einstellungen erstellt wurden. Um die Einstellungen für vorhandene Formulare zu ändern, wählen Sie für die Formulareigenschaften *GridSize*, *DrawGrid* und *SnapToGrid* die entsprechenden Werte.

Siehe auch

Ein erster Blick auf die IDE (siehe Seite 1363)

Ein Projekt starten (siehe Seite 1369)

Komponenten in ein Formular einfügen (siehe Seite 55),

1.50 Symbolleisten anpassen

So ordnen Sie Symbolleisten an:

1. Klicken Sie auf die Griffleiste am linken Rand der Symbolleiste.
2. Ziehen Sie die Symbolleiste mit der Maus an eine andere Position auf dem Desktop.

So entfernen Sie Schaltflächen aus der Symbolleiste:

1. Wählen Sie **Ansicht**▶**Symbolleisten**▶**Anpassen**.
2. Ziehen Sie die Schaltfläche aus der Symbolleiste (nicht aus dem Dialogfeld Anpassen), bis ein X als ihr Symbol angezeigt wird, und lassen Sie dann die Maustaste los.
3. Klicken Sie auf die Schaltfläche Schließen.

So fügen Sie der Symbolleiste Schaltflächen hinzu:

1. Wählen Sie **Ansicht**▶**Symbolleisten**▶**Anpassen**.
2. Klicken Sie auf die Registerkarte Anweisungen.
3. Wählen Sie in der Liste Kategorien eine Kategorie aus, um die darin enthaltenen Schaltflächensymbole anzuzeigen.
4. Ziehen Sie das ausgewählte Schaltflächensymbol aus der Liste Anweisungen in die Symbolleiste Ihrer Wahl.
5. Klicken Sie auf die Schaltfläche Schließen.

Siehe auch

Ein erster Blick auf die IDE (☞ siehe Seite 1363)

Tool-Palette anpassen (☞ siehe Seite 64)

1.51 Tool-Palette anpassen

So ordnen Sie einzelne Komponenten neu an:

1. Klicken Sie auf die Komponente.
2. Ziehen Sie die Komponente an die gewünschte Position auf der Tool-Palette.

So ordnen Sie eine komplette Komponentenkategorie neu an:

1. Klicken Sie auf einen Kategorienamen.
2. Ziehen Sie die Kategorie an die gewünschte Position auf der Tool-Palette.
3. Lassen Sie die Maustaste los, um die Kategorie an der neuen Position zu platzieren.

So fügen Sie neue Kategorien hinzu:

1. Klicken Sie mit der rechten Maustaste auf die Tool-Palette.
2. Wählen Sie Neue Kategorie hinzufügen. Das Dialogfeld Eine neue Kategorie erstellen wird angezeigt.
3. Geben Sie einen Namen für die Kategorie in das Feld Neuer Kategorienname ein.
4. Klicken Sie auf OK. Die neue Kategorie wird nun unten in der Tool-Palette angezeigt.

Siehe auch

Komponenten-Templates erzeugen (↗ siehe Seite 60)

1.52 Deaktivieren von Themes in der IDE und in Ihrer Anwendung

Windows Vista und Windows XP unterstützen Themes in der Benutzeroberfläche. Die IDE verwendet standardmäßig Themes, und Laufzeit-Themes sind für die Anwendung selbst aktiviert. Wenn Sie die klassische Benutzeroberfläche bevorzugen, können Sie die Verwendung von Themes in der IDE und in Ihrer Anwendung deaktivieren..

So deaktivieren Sie Themes für die IDE:

1. Schließen Sie die IDE.
2. Öffnen Sie die Datei `bds.exe.manifest`, die sich im Verzeichnis `$(IDE)\bin` befindet.
3. Entfernen Sie den folgenden Eintrag aus der XML-Datei `bds.exe.manifest` oder kommentieren Sie ihn aus:

4. Speichern Sie die Änderungen der Datei `bds.exe.manifest`.
5. Starten Sie die IDE neu.

So deaktivieren Sie Themes für eine Anwendung:

1. Wählen Sie **Projekt**▸**Optionen**▸**Anwendung**.
2. Deaktivieren Sie das Kontrollkästchen Laufzeit-Themes aktivieren.

Siehe auch

Ein Projekt starten (siehe Seite 1363)

1.53 Tool-Fenster andocken

Mit der Funktion des automatischen Ausblendens können Sie Tool-Fenster wie den Objektinspektor, die Tool-Palette und die Projektverwaltung aus der Verankerung lösen und verbergen, haben aber dennoch Zugriff darauf.

So blenden Sie die Tools automatisch aus:

1. Klicken Sie auf die Pinn-Nadel in der oberen rechten Ecke eines Tool-Fensters. Das Tool-Fenster wird durch eines oder mehrere Register am äußeren Rand des IDE-Fensters ersetzt.
2. Um das Tool-Fenster anzuzeigen, zeigen Sie mit der Maus auf das Register. Das Tool-Fenster wird daraufhin eingeblendet.
3. Um das Tool-Fenster wieder auszublenden, bewegen Sie den Mauszeiger aus dem Tool-Fenster heraus.
4. Um ein Tool-Fenster wieder anzudocken, klicken Sie einfach erneut auf die Pinn-Nadel, bis diese wieder nach unten zeigt.

So docken Sie Tool-Fenster aneinander an:

1. Klicken Sie auf die Titelleiste eines Tool-Fensters und ziehen Sie es in ein anderes Tool-Fenster.
2. Wählen Sie eine Position aus, an der das Tool-Fenster platziert werden soll, und lassen Sie die Maustaste los.

So lösen Sie angedockte Tool-Fenster voneinander:

1. Klicken Sie auf die Titelleiste eines Tool-Fensters und ziehen Sie es aus dem anderen Tool-Fenster heraus.
2. Wählen Sie eine Position aus, an der das Tool-Fenster platziert werden soll, und lassen Sie die Maustaste los.

Siehe auch

Desktop-Layouts speichern (☞ siehe Seite 72)

Voreinstellungen für Tools festlegen (☞ siehe Seite 79)

1.54 Elemente der Tool-Palette suchen

So suchen Sie Elemente der Tool-Palette:

1. Klicken Sie auf eine beliebige Stelle der Tool-Palette, und geben Sie den Namen des zu suchenden Elements ein. Die Tool-Palette wird gefiltert, sodass nur die Elementnamen angezeigt werden, die mit der von Ihnen eingegebenen Zeichenfolge übereinstimmen. Die eingegebenen Zeichen werden in den Elementnamen in **Fettschrift** dargestellt.
2. Doppelklicken Sie auf ein Element, um dessen Standardaktion auszuführen. Bei einem Doppelklick auf eine Komponente wird diese beispielsweise dem Formular hinzugefügt. Wenn Sie dagegen auf ein Code-Snippet doppelklicken, wird dieses in den Quelltext eingefügt.
3. Um den Suchfilter für die Tool-Palette zu entfernen, klicken Sie auf das Filtersymbol .

Siehe auch

Der Tool-Palette Komponenten hinzufügen ( siehe Seite 70)

1.55 Metadaten von .NET-Assemblierungen untersuchen

Sie können die Namespaces und Typen anzeigen, die in einer .NET-Assemblierung enthalten sind. Die Metadaten einer Assemblierung werden in einer Darstellungsform angezeigt, die dem Windows Explorer ähnlich ist: der linke Bereich zeigt die Struktur der Namespaces und Typen in der Assemblierung an. Im rechten Bereich werden jeweils spezifische Informationen zu dem Element angezeigt, das im linken Fensterbereich ausgewählt wurde. Auf der Registerkarte Aufruf-Graph wird sowohl eine Liste derjenigen Methoden angezeigt, die von der aktuell ausgewählten Methode aufgerufen werden, als auch eine Liste derjenigen Methoden, die die ausgewählte Methode aufrufen.

So untersuchen Sie den Inhalt einer .NET-Assemblierung:

1. Wählen Sie **Datei > Öffnen**.
2. Klicken Sie im Dialogfeld Öffnen in der Liste Dateitypen auf den Eintrag **Assemblierungs-Metadaten**.
3. Wechseln Sie zu dem Ordner, in dem sich die .NET-Assemblierung befindet. Wählen Sie die Assemblierung aus, und klicken Sie auf Öffnen.

Sie können im Metadaten-Explorer mehrere .NET-Assemblierungen öffnen. Jede geöffnete Assemblierung wird in der Struktur im linken Fensterbereich angezeigt. Der Knoten für eine .NET-Assemblierung auf der obersten Ebene wird durch das Assemblierungsknoten-Symbol  dargestellt.

Um eine bestimmte .NET-Assemblierung wieder zu schließen, klicken Sie mit der rechten Maustaste auf das Assemblierungsknoten-Symbol  der obersten Ebene und wählen Schließen..

So verwenden Sie den Aufruf-Graph:

1. Wählen Sie im linken Fensterbereich einen Methodenknoten aus.
2. Klicken Sie auf das Register Aufruf-Graph. In der oberen Hälfte der Registerkarte Aufruf-Graph wird eine Liste derjenigen Methoden angezeigt, die die von Ihnen im linken Fensterbereich markierte Methode aufrufen. In der unteren Hälfte der Registerkarte Aufruf-Graph wird eine Liste derjenigen Methoden angezeigt, die von der im linken Fensterbereich markierten Methode aufgerufen werden. Methoden, die sich in derselben Assemblierung befinden wie die aktuell ausgewählte Methode, werden als blau unterstrichene Links angezeigt, die durch Klicken aktiviert werden können. Wenn Sie auf einen Link klicken, wird die betreffende Methode im Baumdiagramm im linken Fensterbereich ausgewählt.

Tip: Sie können die Browser-Schaltflächen in der Symbolleiste dazu verwenden, vorwärts und rückwärts zu navigieren und die bereits ausgewählten Elemente im linken Fensterbereich erneut anzuzeigen.

Siehe auch

Assemblierungs-Metadaten-Explorer (IDE-Referenz) ( siehe Seite 833)

1.56 Windows-Typbibliotheken untersuchen

Sie können die Interfaces und andere Typen öffnen und prüfen, die in der Windows-Typbibliothek enthalten sind. Der Inhalt der Typbibliothek wird im Stil des Windows Explorers dargestellt, wobei der linke Fensterbereich die Struktur des Interface und der Typdefinitionen innerhalb der Typbibliothek anzeigt. Im rechten Bereich werden jeweils spezifische Informationen zu dem Element angezeigt, das im linken Fensterbereich ausgewählt wurde. Mit dem Typbibliothek-Explorer können **.TLB**-Dateien, OCX-Steuerelemente sowie **.DLL**- und **.EXE**-Dateien mit Typbibliotheken als eingebettete Ressourcen geöffnet werden.

So untersuchen Sie eine Windows-Typbibliothek:

1. Wählen Sie **Datei > Öffnen**.
2. Klicken Sie im Dialogfeld Öffnen in der Liste Dateitypen auf den Eintrag Typbibliothek. Dadurch werden lediglich Dateien mit der Namenserweiterung **.TLB**, **.OLB**, **.OCX**, **.DLL** und **.EXE** angezeigt.
3. Wechseln Sie zum Ordner, in dem sich die Typbibliothek befindet.
4. Markieren Sie die Datei, und klicken Sie auf Öffnen.

Sie können im Explorer mehrere Typbibliotheken gleichzeitig öffnen. Jede geöffnete Typbibliothek wird in der Struktur im linken Fensterbereich angezeigt. Der Knoten für eine Typbibliothek auf der obersten Ebene wird durch das Symbol  dargestellt.

Um eine bestimmte Typbibliothek wieder zu schließen, klicken Sie mit der rechten Maustaste auf das Symbol  der obersten Ebene und wählen Schließen.

Siehe auch

Typbibliothek-Explorer (IDE-Referenz) (siehe Seite 835)

1.57 Benutzerdefinierte Komponenten installieren

So installieren Sie benutzerdefinierte Komponenten:

1. Wählen Sie **Komponenten**▶**Installierte .NET-Komponenten**.
2. Klicken Sie auf Wählen Sie eine Assemblierung.
3. Wechseln Sie zum Ordner mit der Komponentenassembly. Alternativ können Sie den Namen und den kompletten Verzeichnispfad der Assemblierung in das Feld Dateiname eingeben.
4. Wählen Sie die Assemblierung aus.
5. Klicken Sie auf Öffnen. Im Dialogfeld Installierte .NET-Komponenten werden die in der Assemblierung enthaltenen Komponenten angezeigt.
6. Stellen Sie sicher, dass die in der Tool-Palette zu installierenden Komponenten aktiviert sind.
7. Klicken Sie auf OK.

Siehe auch

Komponenten in ein Formular einfügen (siehe Seite 55)

1.58 Dateien in der Projektverwaltung umbenennen

Wenn Sie eine Datei umbenennen, wird der Dateiname sowohl in der Projektverwaltung als auch auf der Festplatte geändert.

So benennen Sie eine Datei um:

1. Klicken Sie in der Projektverwaltung mit der rechten Maustaste auf die Datei, die Sie umbenennen möchten. Das Kontextmenü wird angezeigt.
2. Wählen Sie **Umbenennen**.
3. Geben Sie den neuen Namen für die Datei ein. Alle zugehörigen Dateien, die in der Projektverwaltung als untergeordnete Knoten zu sehen sind, werden ebenfalls umbenannt.

Siehe auch

Ein erster Blick auf die IDE (siehe Seite 1363)

1.59 Desktop-Layouts speichern

Sie können zwischen verschiedenen Desktop-Layouts umschalten. Wählen Sie dazu das gewünschte Layout in der Dropdown-Liste der Symbolleiste Desktop aus. Ferner können Sie das Layout des Desktops oder Debug-Desktops als Standard speichern.

So speichern Sie ein Desktop-Layout:

1. Wählen Sie **Ansicht**▸**Desktops**▸**Desktop speichern**.
2. Geben Sie einen Namen für das Desktop-Layout in das Dialogfeld Desktop speichern ein.
3. Klicken Sie auf OK.

So wählen Sie ein Debug-Desktop-Layout aus:

1. Wählen Sie **Ansicht**▸**Desktops**▸**Debug-Desktop einstellen**.
2. Wählen Sie ein Layout für den Debug-Desktop aus.
3. Klicken Sie auf OK.

Siehe auch

Projektoptionen einstellen (☞ siehe Seite 75)

Überblick zum Debuggen (☞ siehe Seite 1439)

1.60 Komponenteneigenschaften festlegen

Nachdem Sie die Komponenten im Designer platziert haben, definieren Sie deren Eigenschaften im Objektinspektor. Durch das Festlegen der Komponenteneigenschaften ändert sich die Darstellung und Funktionsweise einer Komponente in der Anwendung. Da auf die Eigenschaften zur Entwurfszeit zugegriffen werden kann, lassen sich diese jederzeit überprüfen und bei Bedarf ohne weitere Quelltexteingabe auf einfache Weise ändern.

So legen Sie Komponenteneigenschaften fest:

1. Klicken Sie im Objektinspektor auf das Register Eigenschaften.
2. Weisen Sie die Komponenteneigenschaften zu, indem Sie Werte in das Textfeld oder in einen Editor eingeben. Bei Eigenschaften mit booleschen Werten, wie **True** und **False**, können Sie zwischen den zwei Werten wechseln.

Siehe auch

Ein Projekt erstellen (siehe Seite 61)

1.61 Dynamische Eigenschaften festlegen

Viele .NET-Framework-Objekte unterstützen dynamische Eigenschaften. Dynamische Eigenschaften stellen eine Möglichkeit dar, die Werte von Eigenschaften zu ändern, ohne die Anwendung neu compilieren zu müssen. Die dynamischen Eigenschaften und deren Werte werden in einer Konfigurationsdatei gespeichert, die der ausführbaren Datei der Anwendung zugeordnet ist. Die Änderung eines Eigenschaftswerts in der Konfigurationsdatei wird wirksam, wenn die Anwendung das nächste Mal ausgeführt wird. Mithilfe dynamischer Eigenschaften kann eine Anwendung geändert werden, nachdem sie weitergegeben wurde.

So legen Sie eine dynamische Eigenschaft im Objektinspektor fest:

1. Klicken Sie auf der Registerkarte Design eines Formulars auf das Objekt, für das Sie dynamische Eigenschaften festlegen möchten.
2. Erweitern Sie im Objektinspektor den Knoten (DynamicProperties), und klicken Sie auf (Erweitert). Wenn das Objekt keine dynamischen Eigenschaften unterstützt, wird der Knoten (DynamicProperties) nicht angezeigt.

Tip: Falls die Anzeige im Objektinspektor nach Kategorien angeordnet ist, wird (DynamicProperties) unter Konfigurationen angezeigt.

3. Klicken Sie auf die Ellipsenschaltfläche ... neben (Erweitert), um das Dialogfeld Dynamische Eigenschaften zu öffnen. In diesem Dialogfeld sind alle Eigenschaften verzeichnet, die in der Konfigurationsdatei gespeichert werden können.
4. Markieren Sie die Eigenschaften, die Sie in der Konfigurationsdatei speichern möchten.
5. Sie können optional auch den Standardschlüsselnamen ändern, der im Feld Schlüsselzuweisung angezeigt wird.
6. Klicken Sie auf OK. Die dynamischen Eigenschaften werden im Objektinspektor durch ein Symbol gekennzeichnet. RAD Studio erstellt im Projektverzeichnis eine XML-Datei namens `app.config` (für eine Windows-Anwendung) bzw. `web.config` (für eine Web-Anwendung). In dieser Datei sind die dynamischen Eigenschaften und deren aktuelle Werte aufgelistet.
7. Compilieren Sie die Anwendung. RAD Studio erstellt in dem Verzeichnis, in dem die ausführbare Datei bzw. die DLL-Datei der Anwendung abgelegt wird, eine Datei namens `<Projektname>.exe.config` (für eine Windows-Anwendung) bzw. `<Projektname>.dll.config` (für eine Web-Anwendung).

So ändern Sie den Wert einer dynamischen Eigenschaft in der Konfigurationsdatei:

1. Suchen Sie in dem Verzeichnis, das die ausführbare Datei bzw. die DLL-Datei der Anwendung enthält, nach der Konfigurationsdatei.
2. Öffnen Sie die Datei in einem Texteditor.
3. Suchen Sie nach der Anweisung `add key=` für die zu ändernde Eigenschaft, und bearbeiten Sie den Wert.
4. Speichern Sie die Änderungen, und schließen Sie die Datei.

Beim nächsten Start der Anwendung wird der geänderte Eigenschaftswert verwendet.

Siehe auch

[Einführung in dynamische Eigenschaften](#)

1.62 Projektoptionen einstellen

Die Optionen für Anwendung und Compiler lassen sich projektweise definieren. Änderungen an den Projektoptionen betreffen nur das jeweils aktuelle Projekt. Die vorgenommenen Einstellungen lassen sich aber auch als Standardeinstellungen für neue Projekte speichern.

So ändern Sie die Compiler-Optionen:

1. Wählen Sie **Projekt > Optionen**. Das Dialogfeld Optionen wird geöffnet.
2. Wählen Sie Compiler, und stellen Sie die Optionen ein, die Sie für die Compilierung des Programms verwenden möchten.
3. Klicken Sie auf OK.

So ändern Sie die Anwendungsoptionen:

1. Wählen Sie **Projekt > Optionen**. Das Dialogfeld Optionen wird geöffnet.
2. Wählen Sie Anwendung, und geben Sie einen Namen und eine Erweiterung für Ihre Anwendung ein.
3. Klicken Sie auf OK.

So ändern Sie die Debugger-Optionen:

1. Wählen Sie **Projekt > Optionen**. Das Dialogfeld Optionen wird geöffnet.
2. Verwenden Sie die Seite Debugger zur Übergabe von Befehlszeilenparametern an die Anwendung, zur Festlegung einer Host-Anwendung für den Test einer DLL oder zum Laden einer ausführbaren Datei in den Debugger.
3. Verwenden Sie die Seite Umgebung zur Angabe der Umgebungsvariablen, die während des Debuggens an die Anwendung übergeben werden.
4. Klicken Sie auf OK.

Siehe auch

- Ein erster Blick auf die IDE ([siehe Seite 1363](#))
- Komponenten in ein Formular einfügen ([siehe Seite 55](#))
- Dateien hinzufügen und entfernen ([siehe Seite 57](#))
- Ein Projekt erstellen ([siehe Seite 61](#))
- Eigenschaften ([siehe Seite 77](#))

1.63 Festlegen von C++-Standardprojektoptionen

Die Optionen für Anwendung und Compiler lassen sich projektweise definieren. Änderungen an den Projektoptionen betreffen nur das jeweils aktuelle Projekt. Die vorgenommenen Einstellungen lassen sich aber auch als Standardeinstellungen für neue Projekte speichern.

So ändern Sie Optionswerte:

1. Wählen Sie **Projekt > Optionen**. Das Dialogfeld Optionen wird geöffnet.
2. Wählen Sie im linken Bereich eine Seite aus der Liste aus.
3. Wenn Sie den Cursor auf einen Optionsnamen setzen, wird ein Kurzhinweis angezeigt, der eine Optionsbeschreibung, den Standardwert der Option und die Befehlszeilenoption (falls vorhanden) enthält.
4. Wenn der Wert einer Option von dem Wert der übergeordneten Konfiguration abweicht, wird der zugehörige Text in Fettschrift dargestellt.
5. Setzen Sie Optionen auf der Seite, um festzulegen, wie Ihr Projekt erzeugt werden soll. Abhängig von der jeweiligen Option können Sie Text eingeben, ein Kontrollfeld markieren oder demarkieren oder eine Auswahl in einem Pulldown-Menü treffen. Einige Optionen verfügen über eine Ellipsen-Schaltfläche, die nach dem Anklicken ein Dialogfeld zum Auswählen einer Datei oder eines Verzeichnisses oder zum Verwalten einer Liste mit Einträgen, wie etwa eine Pfadliste, anzeigen.
6. Neben einigen Optionen, die eine Liste mit Einträgen enthalten, wie z.B. Definitionen oder Pfadangaben, wird das Kontrollfeld Zusammenführen angezeigt. Wenn dieses Kontrollfeld aktiviert ist, führt die IDE die Optionsliste mit der des unmittelbaren Vorfahren der Konfigurationsliste für diese Option zusammen. Beachten Sie, dass die IDE den Inhalt der Option nicht eigentlich ändert, sondern sich so verhält, als ob die Liste die Liste des Vorfahren enthielte. Wenn das Kontrollfeld Zusammenführen des Vorfahren auch aktiviert ist, führt die IDE auch die Liste dieses Vorfahren für diese Option zusammen, und weiter die Vererbungskette hinauf. Ist das Kontrollfeld deaktiviert, verwendet die IDE nur die Einträge aus der aktuellen Konfiguration.
7. Klicken Sie auf OK, um die Änderungen zu übernehmen und das Dialogfeld zu schließen. Klicken Sie auf OK, um die Änderungen zu ignorieren und das Dialogfeld zu schließen.

So setzen Sie Optionswerte auf die Werte der übergeordneten Konfiguration zurück:

1. Wählen Sie **Projekt > Optionen**. Das Dialogfeld Optionen wird geöffnet.
2. Wählen Sie aus der Liste im linken Bereich eine Seite aus, und setzen Sie die Optionen, um festzulegen, wie Ihr Projekt erzeugt werden soll.
3. Wenn der Wert einer Option von dem Wert der übergeordneten Konfiguration abweicht, wird der zugehörige Text in Fettschrift dargestellt.
4. Klicken Sie mit der rechten Maustaste auf die Option, und wählen Sie aus dem Kontextmenü Zurücksetzen. Der Optionswert wird in den der übergeordneten Konfiguration geändert.
5. Klicken Sie auf OK, um die Änderungen zu übernehmen und das Dialogfeld zu schließen. Klicken Sie auf OK, um die Änderungen zu ignorieren und das Dialogfeld zu schließen.

Siehe auch

- Ein erster Blick auf die IDE (siehe Seite 1363)
- Komponenten in ein Formular einfügen (siehe Seite 55)
- Dateien hinzufügen und entfernen (siehe Seite 57)
- Ein Projekt erstellen (siehe Seite 61)
- Eigenschaften (siehe Seite 77)

1.64 Eigenschaften und Ereignisse festlegen

Eigenschaften, Methoden und Ereignisse sind Attribute einer Komponente.

So legen Sie Objekteigenschaften fest:

1. Klicken Sie im Formular auf ein Objekt, um es auszuwählen.
2. Klicken Sie im Objektinspektor auf das Register Eigenschaften.
3. Wählen Sie die Eigenschaft aus, die Sie ändern möchten, und geben Sie dann den Wert in das Textfeld ein, oder klicken Sie auf die Ellipsen-Schaltfläche (...) und geben den Wert über den entsprechenden Eigenschaftseditor an.

So legen Sie eine Ereignisbehandlungsroutine fest:

1. Klicken Sie im Formular auf ein Objekt, um es auszuwählen.
2. Klicken Sie im Objektinspektor auf das Register Ereignisse.
3. Gibt es für dieses Objekt bereits eine vordefinierte Ereignisbehandlungsroutine, wählen Sie diese in der Dropdown-Liste aus. Andernfalls doppelklicken Sie auf das Ereignis, um in die Ansicht Quelltext zu wechseln.
4. Geben Sie den Code ein, der ausgeführt werden soll, sobald das Ereignis eintritt.

Siehe auch

Ein erster Blick auf die IDE (siehe Seite 1363)

Komponenten in ein Formular einfügen (siehe Seite 55)

Dateien hinzufügen und entfernen (siehe Seite 57)

Ein Projekt erstellen (siehe Seite 61)

Projektoptionen einstellen (siehe Seite 75)

1.65 Die IDE für das Simulieren von Delphi 7 einrichten

Mit diesem Verfahren können Sie die IDE zum Simulieren von Delphi 7 oder C++ Builder einrichten, wobei jeder Bereich ein eigenes Fenster besitzt.

So deaktivieren Sie den eingebetteten Designer:

1. Wählen Sie **Tools**▶**Optionen**▶**Umgebungsoptionen**▶**VCL-Designer**.
2. Deaktivieren Sie Eingebetteter Designer.
3. Klicken Sie auf OK.
4. Starten Sie RAD Studio neu, damit die Änderung wirksam wird.

1.66 Voreinstellungen für Tools festlegen

Sie können viele Aspekte im Erscheinungsbild und Verhalten von Tools und Funktionen, wie dem Objektinspektor, dem Quelltext-Editor und dem integrierten Debugger, anpassen.

So legen Sie die Voreinstellungen für Tools fest:

1. Wählen Sie **Tools > Optionen**.
2. Überprüfen Sie die Optionen in jeder Tool-Kategorie, und ändern Sie die Einstellungen entsprechend Ihren Erfordernissen.
3. Klicken Sie auf OK.

Siehe auch

Formular anpassen ( siehe Seite 62)

Tool-Palette anpassen ( siehe Seite 64)

Projektoptionen einstellen ( siehe Seite 75)

1.67 Designer-Richtlinien für VCL-Komponenten verwenden

Mit VCL und VCL.NET (Delphi oder C++) können Sie Komponenten so definieren, dass sie sich "relativ" zu den anderen Komponenten im Formular verhalten. Mit Hilfe entsprechender Eigenschaften lässt sich der Abstand zwischen Steuerelementen, Verknüpfungen, Fokusbeschriftungen, der Tabulatorreihenfolge und den Einträgen von Listenfeldern und Menüs festlegen.

So können Sie Designer-Richtlinien anzeigen und verwenden:

1. Registrieren Sie einen Objekttyp.
2. Definieren Sie Stellen auf oder in der Nähe der Grenze einer Komponente, die als Ausrichtungspunkte fungieren sollen. Die Ausrichtungspunkte werden durch vertikale oder horizontale Linien definiert, die die Grenzen eines visuellen Steuerelements schneiden.
3. Definieren Sie Oberflächen-Richtlinien für Komponenten. Sie können z.B. Richtlinien vorgeben, die den Abstand zwischen Steuerelementen, Verknüpfungen, Fokusbeschriftungen, der Tabulatorreihenfolge und den Einträgen von Listenfeldern und Menüs festlegen.

Im oberen Bereich des neuen Dialogfelds zur Fehlerbehebung werden vier Spalten angezeigt, im unteren Bereich stehen sechs Optionsfelder zu Wahl. Die folgende Tabelle enthält eine Beschreibung der Spalten.

Komponente	Komponente Vorgabewert bei aktivierter Option "Designer-Richtlinien verwenden"
Ausrichtung	Die Namen der Tabellenspalten, in denen ein Fehler aufgetreten ist.
Ränder	Unten = 3, Links = 3, Rechts = 3, Rechts = 3, Oben = 3
Padding	Die letzte auf dem Server gespeicherte Aktualisierung. (Entspricht dem Inhalt der Zeile auf dem Server.)

Siehe auch

Benutzeroberflächen entwerfen (siehe Seite 1410)

Ein erster Blick auf die IDE (siehe Seite 1363)

1.68 Verwenden des Datei-Browsers

Der Datei-Browser ist ein Browser im Windows-Stil, den Sie in der IDE andocken können. Mithilfe des Kontextmenüs des Datei-Browsers lassen sich Dateioperationen, wie z.B. Ausschneiden, Kopieren, Löschen und Umbenennen, ausführen. Mit dem Befehl Dem Projekt hinzufügen können Sie Ihrem Projekt auch eine Datei hinzufügen.

So öffnen und verwenden Sie den Datei-Browser:

1. Wählen Sie in der IDE **Ansicht**►**Datei-Browser**.
2. Klicken Sie mit der rechten Maustaste auf einen Dateinamen, und wählen Sie den gewünschten Befehl aus dem Kontextmenü aus. Sie können mit RAD Studio Dateien öffnen, ausgewählte Dateien zum aktuellen Projekt hinzufügen oder Windows-Standardoperationen mit Dateien ausführen.

So filtern Sie die Dateiliste:

1. Klicken Sie in der Menüleiste des Datei-Browsers auf die Schaltfläche Filter.
2. Geben Sie im Dialogfeld Filter setzen Dateinamen oder Platzhalterausdrücke getrennt durch Semikolon ein. Der Datei-Browser zeigt alle Dateien an, die den Platzhalterausdrücken der von Ihnen eingegebenen Filterkriterien entsprechen.

Siehe auch

Datei-Browser (☞ siehe Seite 806)

1.69 To-Do-Listen verwenden

In einer To-Do-Liste werden die Programmierarbeiten eingetragen und angezeigt, die in einem Projekt noch ausgeführt werden müssen.

So erstellen Sie eine To-Do-Liste und fügen einen Eintrag hinzu:

1. Wählen Sie [Ansicht>To-Do-Liste](#).
2. Klicken Sie im Dialogfeld To-Do-Liste mit der rechten Maustaste, und wählen Sie Hinzufügen.
3. Geben Sie im Dialogfeld To-Do-Eintrag hinzufügen eine Beschreibung der Aufgabe ein, und passen die anderen Felder nach Bedarf an.
4. Klicken Sie auf OK.

So fügen Sie einen To-Do-Listeneintrag als Kommentar in den Quelltext ein:

1. Setzen Sie den Cursor im Quelltext-Editor an die Position, an der Sie den Kommentar hinzufügen möchten.
2. Klicken Sie mit der rechten Maustaste, und wählen Sie [To-Do-Eintrag hinzufügen](#).
3. Wählen Sie im Dialogfeld To-Do-Eintrag hinzufügen den Eintrag aus, den Sie einfügen möchten.
4. Klicken Sie auf OK.

Der Eintrag wird als Kommentar in den Quelltext eingefügt, wobei ihm das Wort [TODO](#) vorangestellt wird.

So markieren Sie einen To-Do-Listeneintrag als erledigt:

1. Wählen Sie [Ansicht>To-Do-Liste](#).
2. Klicken Sie im Dialogfeld To-Do-Liste auf das Kontrollkästchen neben dem Eintrag, der als erledigt markiert werden soll. Der Eintrag bleibt in der Liste, wird aber mit durchgestrichenem Text angezeigt. Wenn der Eintrag dem Quelltext als Kommentar hinzugefügt wurde, wird er nun mit [DONE](#) statt mit [TODO](#) angezeigt.

So filtern Sie die Einträge einer To-Do-Liste:

1. Wählen Sie [Ansicht>To-Do-Liste](#).
2. Klicken Sie im Dialogfeld To-Do-Liste mit der rechten Maustaste, und wählen Sie Filter.
3. Wählen Sie je nach gewünschtem Filterkriterium entweder Kategorien, Besitzer oder Eintragstypen aus.
4. Deaktivieren Sie im Dialogfeld To-Do-Liste filtern die Kontrollkästchen derjenigen Einträge, die in der To-Do-Liste ausgeblendet werden sollen.
5. Klicken Sie auf OK. Die To-Do-Liste wird wieder angezeigt, und die gefilterten Einträge sind nicht mehr zu sehen. Die Statusleiste unten im Dialogfeld To-Do-Liste zeigt an, wie viele Einträge aufgrund des aktiven Filters verborgen werden.

So löschen Sie einen Eintrag aus der To-Do-Liste:

1. Wählen Sie [Ansicht>To-Do-Liste](#).
2. Markieren Sie im Dialogfeld To-Do-Liste den zu löschenen Eintrag.
3. Klicken Sie mit der rechten Maustaste, und wählen Sie Löschen. Der Eintrag wird aus der To-Do-Liste gelöscht. Falls der Eintrag als Kommentar in den Quelltext eingefügt wurde, wird auch der betreffende Kommentar gelöscht.

1.70 Verwenden von virtuellen Ordnern

Die IDE ermöglicht die Organisation Ihrer Projektdateien in der Baumstruktur der Projektverwaltung mit Hilfe von virtuellen Ordnern. Virtuelle Ordner wirken sich nur auf die Anzeige der Ordnerstruktur in der IDE aus. Das Verlagern von Dateien in virtuelle Ordner verändert deren Speicherort auf dem Datenträger nicht.

Anmerkung: Virtuelle Ordner dürfen nur Dateisystemeinträge und andere virtuelle Ordner enthalten.

Anmerkung: Das Ändern der Reihenfolge der Einträge in einem virtuellen Ordner ändert die Build-Reihenfolge der enthaltenen Einträge.

So fügen Sie der Stammebene einen virtuellen Ordner hinzu:

1. Wählen Sie **Ansicht**▶**Projektverwaltung**, um die Projektverwaltung zu öffnen, falls sie nicht bereits in der IDE angezeigt wird.
2. Erstellen Sie ein neues Projekt oder öffnen Sie ein bestehendes.
3. Klicken Sie den Projektknoten in der Baumstruktur der Projektverwaltung mit der rechten Maustaste an, und wählen Sie **Neue hinzufügen**▶**Virtueller Ordner**. Das Dialogfeld Neuen Ordner hinzufügen mit dem Standardordnernamen **Virtueller Ordner 1** wird geöffnet.
4. Wenn Sie den Standardnamen nicht verwenden möchten, geben Sie einen neuen Namen für den virtuellen Ordner ein.
5. Klicken Sie auf OK, um den Ordner hinzuzufügen. Klicken Sie auf Abbrechen, um den Ordner nicht hinzuzufügen. Der virtuelle Ordner wird als abgedunkelter Ordner unter dem Projektknoten angezeigt.
6. Ziehen Sie aus der Projektstruktur Dateien in den virtuellen Ordner, oder fügen Sie mit den Befehlen des Kontextmenüs Elemente in den virtuellen Ordner ein.

So fügen Sie einem vorhandenen virtuellen Ordner virtuelle Unterordner hinzu:

1. Klicken Sie in der Baumstruktur der Projektverwaltung einen vorhandenen virtuellen Ordner mit der rechten Maustaste an.
2. Wählen Sie **Neue hinzufügen**▶**Virtueller Ordner**.

So ändern Sie in einem virtuellen Ordner die Reihenfolge von Dateien:

1. Klicken Sie auf eine Datei in einem virtuellen Ordner und ziehen Sie die Datei. Der Cursor wird als Pfeil mit einem Rechteck angezeigt, um anzugeben, dass Sie die Datei verschieben.
2. Ziehen Sie die Datei nach rechts, bis eine horizontale blaue Linien erscheint. Diese Linie zeigt die neue Position der Datei in dem virtuellen Ordner und erscheint nur zwischen Dateien. Wenn der Cursor für eine Position als durchgestrichener Kreis angezeigt wird, können Sie die Datei nicht dort ablegen.
3. Ziehen Sie die Datei an die gewünschte Position in dem virtuellen Ordner und lassen Sie die Maustaste los.

So löschen Sie einen virtuellen Ordner:

1. Wählen Sie in der Baumstruktur der Projektverwaltung einen virtuellen Ordner aus.
2. Klicken Sie mit der rechten Maustaste, und wählen Sie Löschen.
3. Klicken Sie im Dialogfeld Bestätigen auf Ja, um den Ordner zu löschen. Klicken Sie auf Nein, oder drücken Sie die Taste ESC, um das Löschen abzubrechen.

Anmerkung: Das Löschen eines virtuellen Ordners löscht keine seiner Dateien auf dem Datenträger. Es wird nur der Ordner aus der Baumstruktur der Projektverwaltung entfernt. Die Dateien verbleiben an ihren ursprünglichen Positionen.

Siehe auch

Überblick über virtuelle Ordner

Dateien hinzufügen und entfernen ( siehe Seite 57)

1.71 Ereignisbehandlungs routinen schreiben

Programmcode reagiert in der Regel auf Ereignisse, die zur Laufzeit bei einer Komponente eintreten, z.B. wenn ein Benutzer auf eine Schaltfläche klickt oder einen Menübefehl auswählt. Der Quelltext, der auf ein solches Ereignis reagiert, wird als Ereignisbehandlungsroutine bezeichnet. Die Ereignisbehandlungsroutine ändert Eigenschaftswerte und ruft Methoden auf.

So geben Sie eine Ereignisbehandlungsroutine ein:

1. Klicken Sie im Formular auf die Komponente, für die Sie eine Ereignisbehandlungsroutine schreiben möchten.
2. Um eine Routine für das Standardereignis der Komponente einzugeben, doppelklicken Sie im Formular auf die Komponente. Zur Auswahl eines anderen Ereignisses für die Komponente klicken Sie im Objektinspektor auf das Register Ereignisse, suchen das gewünschte Ereignis und doppelklicken auf dessen Textfeld. Der Quelltext-Editor wird geöffnet.
3. Geben Sie den Quelltext ein, der ausgeführt werden soll, wenn das Ereignis zur Laufzeit auftritt.

Siehe auch

Ein Projekt erstellen (siehe Seite 61)

Komponenteneigenschaften festlegen (siehe Seite 73)

1.72 Sprachen zu einem Projekt hinzufügen

Mit dem Satellite-Assemblierungsexperten (.NET) bzw. dem Ressourcen-DLL-Experten (Win32) können Sie einem Projekt Sprachen hinzufügen. Für jede hinzugefügte Sprache erzeugt der Experte in der zugehörigen Projektgruppe ein Ressourcenmodul. Die verschiedenen Ressourcenmodul-Projekte erhalten jeweils eine vom Gebietsschema der Sprache abhängige Erweiterung.

So fügen Sie einem Projekt eine Sprache hinzu:

1. Speichern und compilieren Sie das Projekt.
2. Vergewissern Sie sich davon, dass das Projekt in der IDE geöffnet ist, und wählen Sie **Projekt**▶**Sprachen**▶**Hinzufügen**. Wahlweise können Sie auch **Datei**▶**Neu**▶**Weitere**▶**Delphi für .NET-Projekte**▶**Satellite-Assemblierungsexperte** (.NET-Anwendung) oder **Datei**▶**Neu**▶**Weitere**▶**Delphi-Projekte**▶**Ressourcen-DLL-Experte** (Win32-Anwendung) wählen. Der Experte wird daraufhin angezeigt.
3. Vergewissern Sie sich, dass Ihr Projekt in der angezeigten Liste ausgewählt ist, und klicken Sie dann auf **Weiter**.
4. Aktivieren Sie das Kontrollkästchen neben der Sprache, die dem Projekt hinzugefügt werden soll, und klicken Sie auf **Weiter**.
5. Überprüfen Sie den vorgeschlagenen Verzeichnispfad für das Ressourcenmodul der Sprache.

Tip: Um den Pfad zu ändern, klicken Sie zuerst auf die Pfadangabe und dann auf die auf die Ellipsen-Schaltfläche (...), und wählen Sie das neue Verzeichnis aus.

Wenn die gewünschte Pfadangabe angezeigt wird, klicken Sie auf **Weiter**.

6. Falls für die Sprache noch keine Satellite-Assemblierung vorhanden ist, wird in der Spalte Aktualisierungsmodus der Eintrag **Neu erstellen** angezeigt. Klicken Sie auf **Weiter**. Falls das angegebene Verzeichnis ein Ressourcenmodul für die Sprache enthält, klicken Sie in die Spalte Aktualisierungsmodus und wählen **Aktualisieren** oder **Überschreiben**. Wählen Sie **Aktualisieren**, wenn Sie das vorhandene Satellite-Assemblierungsprojekt beibehalten und modifizieren möchten. Wählen Sie **Überschreiben**, um ein neues leeres Projekt zu erstellen und das alte Projekt samt allen darin enthaltenen Übersetzungen zu löschen. Klicken Sie auf **Weiter**.
7. Überprüfen Sie die vom Experten angezeigte Zusammenfassung und klicken Sie auf **Fertig stellen**, um die Ressourcenmodule für die von Ihnen gewählten Sprachen zu erstellen oder zu aktualisieren. Wenn der Experte fragt, ob eine **.drcil**-Datei (.NET) oder eine **.drc**-Datei (Win32) erzeugt werden soll, klicken Sie auf **Ja**. In jedem Projekt, das über eigene String-Ressourcen verfügt (und nicht die vorcompilierten **.rc**-Dateien verwendet), muss eine **.drcil**- bzw. **.drc**-Datei vorhanden sein. Wenn Sie sicher sind, dass keine neuen Dateien erforderlich sind (weil das Projekt keine eigenen String-Ressourcen besitzt), wählen Sie im letzten Dialogfeld die Option **Nicht gefundene DRC-(DRCIL)-Dateien überspringen**. Damit wird verhindert, dass der Experte Dateien erzeugt bzw. fragt, ob Dateien erzeugt werden sollen.
8. Klicken Sie auf **Ja**, um das Projekt zu compilieren. Klicken Sie auf **OK**, um die Projektgruppe zu speichern.

Die erzeugten Projekte enthalten unübersetzte Kopien der String-Ressourcen des Originalprojekts. Der Translation-Manager wird automatisch angezeigt, sodass Sie mit dem Übersetzen der Ressourcendateien beginnen können.

So entfernen Sie eine Sprache aus dem aktuellen Projekt:

1. Öffnen Sie ein Projekt.
2. Wählen Sie **Projekt**▶**Sprachen**▶**Entfernen**.
3. Markieren Sie die Sprache(n), die entfernt werden soll(en), und klicken Sie dann auf **Weiter**.
4. Klicken Sie auf **Fertig stellen**.

Der Experte entfernt das gewählten Ressourcenmodul aus der Projektdatei, löscht aber weder die Assemblierungen noch den Quelltext dieser Assemblierungen oder die Verzeichnisse, in denen sich diese befinden.

So fügen Sie eine Sprache erneut in ein Projekt ein:

1. Wählen Sie **Projekt**▸**Sprachen**▸**Hinzufügen**, um den Satellite-Assemblierungsexperten bzw. den Ressourcen-DLL-Experten zu starten.
2. Geben Sie im entsprechenden Dialogfeld den Pfad des zuvor erstellten Ressourcenmoduls an.
3. Wählen Sie in der Spalte Aktualisierungsmodus die Option Aktualisieren. Falls das angegebene Verzeichnis bereits ein Ressourcenmodul für die Sprache enthält, klicken Sie in die Spalte Aktualisierungsmodus und wählen Aktualisieren oder Überschreiben. Wählen Sie Aktualisieren, wenn Sie das vorhandene Assemblierungsprojekt beibehalten und modifizieren möchten. Wählen Sie Überschreiben, um ein neues leeres Projekt zu erstellen und das alte Projekt samt allen darin enthaltenen Übersetzungen zu löschen.
4. Klicken Sie auf Fertig stellen.

Siehe auch

Anwendungen lokalisieren (↗ siehe Seite 1437)

Ressourcendateien im Translation-Manager bearbeiten (↗ siehe Seite 88)

Externen Translation-Manager einrichten (↗ siehe Seite 91)

1.73 Ressourcendateien im Translation-Manager bearbeiten

Nachdem Sie mit dem Satellite-Assemblierungsexperten oder dem Ressourcen-DLL-Experten einem Projekt Sprachen hinzugefügt haben, können Sie die Ressourcendateien mit dem Translation-Manager betrachten und bearbeiten. Sie können Ressourcen-Strings direkt bearbeiten, übersetzte Strings dem Übersetzungswörterbuch hinzufügen und Strings aus dem Übersetzungswörterbuch abrufen.

So bearbeiten Sie Ressourcen-Strings:

1. Öffnen Sie ein Projekt, das Sprachen enthält.
2. Wählen Sie **Ansicht**▸**Translation-Manager**▸**Übersetzungsseditor**.
3. Erweitern Sie die Baumansicht des Projekts, sodass die zu bearbeitenden Ressourcendateien angezeigt werden.

Tip: Verwenden Sie hierzu die Symbole zum Ein- und Ausblenden in der Symbolleiste über der Baumansicht.

4. Klicken Sie auf die Ressourcendatei, die Sie bearbeiten möchten. Die in der Datei enthaltenen Ressourcen-Strings werden in einer Tabelle im rechten Bereich angezeigt.
5. Klicken Sie in das zu bearbeitende Feld und geben den neuen Text direkt in die Tabelle ein, klicken Sie mit der rechten Maustaste auf das Feld und wählen Bearbeiten, um den String in einem Dialogfeld zu bearbeiten, oder klicken Sie in der Symbolleiste über der Tabelle auf das Symbol **Mehrzeilen-Editor**.
6. Sie können bei Bedarf einen Kommentar in das Feld Kommentar eingeben.
7. Optional können Sie über die Dropdown-Liste im Feld Status den Übersetzungsstatus des Strings festlegen.
8. Klicken Sie in der Symbolleiste über der Tabelle auf das Symbol Übersetzung speichern, um die Ressourcendatei zu aktualisieren.

Tip: Sie können das ursprüngliche Formular oder das übersetzte Formular anzeigen, indem Sie in der Symbolleiste über der Tabelle auf das Symbol Originalformular anzeigen bzw. Übersetztes Formular anzeigen klicken.

So fügen Sie einen Ressourcen-String dem Übersetzungswörterbuch hinzu:

1. Klicken Sie nach der Bearbeitung einer Ressourcen-Strings im Translation-Manager mit der rechten Maustaste auf den String, den Sie dem Übersetzungswörterbuch hinzufügen möchten.
2. Wählen Sie **Übersetzungswörterbuch**▸**Übersetzungen dem Wörterbuch hinzufügen**. Der Ressourcen-String wird dem Übersetzungswörterbuch hinzugefügt. Sie können ihn anzeigen, indem Sie den Translation-Manager schließen und **Tools**▸**Übersetzungswörterbuch** wählen.

So rufen Sie einen Ressourcen-String aus dem Übersetzungswörterbuch ab:

1. Klicken Sie im Translation-Manager auf die Registerkarte Arbeitsbereich.
2. Erweitern Sie die Baumansicht des Projekts, sodass die zu bearbeitenden Ressourcendateien angezeigt werden. Die **.resx**-Dateien werden unter dem Knoten **.NET-Ressourcen** aufgeführt. Die **.nfm**-Dateien werden unter dem Knoten **Formulare** angezeigt.
3. Klicken Sie auf die Ressourcendatei, die Sie bearbeiten möchten. Die in der Datei enthaltenen Ressourcen-Strings werden in einer Tabelle im rechten Bereich angezeigt.
4. Klicken Sie mit der rechten Maustaste auf das Feld, das Sie aktualisieren möchten, und wählen Sie **Übersetzungswörterbuch**▸**Strings aus dem Wörterbuch holen**. Falls das Übersetzungswörterbuch nur eine Übersetzung für den gewählten Ausgangs-String enthält, dann wird diese Übersetzung in die Spalte für die Zielsprache kopiert. Enthält das

Wörterbuch mehrere Übersetzungen für die gewählte Ressource, dann wird per Voreinstellung die erste passende Übersetzung abgerufen, die gefunden wird.

Tip: Um diese Voreinstellung zu ändern, schließen Sie den Translation-Manager, wählen **Tools** > **Optionen für Übersetzungs-Tools**, klicken auf die Registerkarte Übersetzungswörterbuch und ändern die Einstellung für Mehrfachsuche.

So öffnen Sie die Ressourcendatei in einem Texteditor:

1. Klicken Sie im Translation-Manager auf die Registerkarte Projekt.
2. Klicken Sie auf die Registerkarte Dateien.
3. Klicken Sie auf die Ressourcendatei, die Sie ändern möchten. Die Datei wird in einem Texteditor geöffnet.
4. Nehmen Sie die gewünschten Änderungen vor und speichern Sie die Datei.

Tip: Sie können den vom Translation-Manager verwendeten Texteditor ändern, indem Sie **Tools** > **Optionen für Übersetzungs-Tools** auswählen und die im Feld Externer Editor angegebene ausführbare Datei ändern.

Siehe auch

Anwendungen lokalisieren (siehe Seite 1437)

Sprachen zu einem Projekt hinzufügen (siehe Seite 86)

1.74 Aktive Sprache für ein Projekt festlegen

Nachdem Sie Ihrem Projekt mit dem Satellite-Assemblierungsexperten oder dem Ressourcen-DLL-Experten Sprachen hinzugefügt haben, wird bei Auswahl von **Start>Ausführen** das Modul für die Basissprache geladen. Sie können jedoch das Laden eines anderen Moduls veranlassen, indem Sie die aktive Sprache für das Projekt festlegen.

So legen Sie die aktive Sprache fest:

1. Compilieren Sie das Ressourcenmodul für die zu verwendende Sprache in der IDE neu.
2. Wählen Sie **Projekt>Sprachen>Als aktiv festlegen**. Der Experte Aktive Sprache festlegen wird gestartet, und eine Liste der im Projekt verwendeten Sprachen wird angezeigt. Die Basissprache ist ganz oben in der Liste in spitzen Klammern angegeben, z.B. `<Englisch (USA)>`.
3. Wählen Sie die gewünschte Sprache in der Liste aus, und klicken Sie auf Fertig stellen.

Siehe auch

Anwendungen lokalisieren (↗ siehe Seite 1437)

Sprachen zu einem Projekt hinzufügen (↗ siehe Seite 86)

1.75 Externen Translation-Manager einrichten

Wenn Ihnen die RAD Studio-IDE nicht zur Verfügung steht, können Sie Anwendungen mit dem externen Translation-Manager (ETM) lokalisieren. Damit Sie den ETM einsetzen können, muss der Entwickler Ihnen die erforderlichen ETM- und Projektdateien zur Verfügung stellen.

Anmerkung: Auf dem Computer, auf dem der ETM installiert werden soll, muss das Microsoft .NET Framework bereits installiert sein.

So konfigurieren und registrieren Sie die ETM-Dateien:

1. Besorgen Sie sich vom Entwickler folgende ETM-Dateien. Diese Dateien befinden sich entweder im Verzeichnis [Programme\CodeGear\RAD Studio\5.0\Bin](#) oder [Windows\system32](#) auf dem Computer des Entwicklers.

Anmerkung: Falls der Entwickler nur die Delphi für Win32-Personality von RAD Studio installiert hat, sind die mit einem Sternchen (*) markierten Dateien nicht vorhanden.

```
Borland.Delphi.dll *
Borland.Globalization.dll *
Borland.ITE.dll *
Borland.ITE.FormDesigner.dll *
Borland.SCI.dll *
Borland.Vcl.dll *
Borland.VclRtl.dll *
Borland.VclX.dll *
designide90.bpl
dfm90.bpl
DotnetCoreAssemblies90.bpl *
etm.exe
IDECtrls90.bpl
itecore90.bpl
itedotnet90.bpl *
rc90.bpl
ResX90.bpl *
rtl90.bpl
vcl90.bpl
vclactnband90.bpl
vclide90.bpl
vclx90.bpl
xmlrtl90.bpl
nfmrtl90.bpl *
```

2. Erstellen Sie ein Verzeichnis, z.B. [C:\ETM](#).
 3. Kopieren Sie die ETM-Dateien, die Sie vom Entwickler erhalten haben, in dieses Verzeichnis.
 4. Starten Sie den externen Translation-Manager. Doppelklicken Sie dazu im Windows-Explorer auf die Anwendung [etm.exe](#), oder geben Sie in der Befehlszeile [etm.exe](#) ein.
 5. Wählen Sie [Tools>Optionen>Packages](#).
 6. Klicken Sie auf die Schaltfläche Hinzufügen, um das Dialogfeld Öffnen anzuzeigen.
 7. Wechseln Sie zum Verzeichnis mit den ETM-Dateien. Vergewissern Sie sich, dass Entwurf-Packages (dcl*.bpl) als Dateityp ausgewählt ist.
 8. Wählen Sie alle Entwurfszeit-Packages im Verzeichnis aus, und klicken Sie auf OK.
- Die Entwurfszeit-Packages sind nun registriert, und Sie können den ETM verwenden.

So richten Sie das zu übersetzende Projekt ein:

1. Fordern Sie beim Entwickler ein Übersetzungskit (als Zip-Datei) für das zu übersetzende Projekt an. Das Übersetzungskit muss Folgendes enthalten:
 - Eine Satellite-Assemblierung oder Ressourcen-DLL für jede Übersetzungssprache
 - Die `.bdsproj`-Projektdatei, die im ETM-Projekt mit **Datei > Speichern unter** erstellt wurden
 - Die eigenständige Wörterbuchdateien (`*.tmx`)
2. Entpacken Sie das Übersetzungskit in einem Verzeichnis Ihrer Wahl.

Siehe auch

Anwendungen lokalisieren ( [siehe Seite 1437](#))

Sprachen zu einem Projekt hinzufügen ( [siehe Seite 86](#))

Ressourcendateien im Translation-Manager bearbeiten ( [siehe Seite 88](#))

1.76 Ressourcenmodule aktualisieren

Wenn Sie eine zusätzliche Ressource (z.B. eine Schaltfläche) in ein Formular einfügen, müssen die Ressourcenmodule aktualisiert werden, um die Änderung zu übernehmen.

So aktualisieren Sie Ressourcenmodule:

1. Speichern und compilieren Sie das Projekt. Wenn Sie mit dem ETM arbeiten, öffnen Sie das gespeicherte Objekt erneut.
2. Aktualisieren Sie die Ressourcenmodule:
 - Wählen Sie in der IDE **Projekt**▸**Sprachen**▸**Lokalisierte Projekte aktualisieren**.
 - Wählen Sie im ETM **Projekt**▸**Aktualisierung ausführen** (oder drücken Sie F9), oder öffnen Sie die Registerkarte Dateien, und klicken Sie auf die Schaltfläche Aktualisierung ausführen (F9).
3. Compilieren Sie nach der Aktualisierung im internen Translation-Manager alle Ressourcenmodule neu. Öffnen Sie dazu das Projekt in der IDE, und wählen Sie **Projekt**▸**Compilieren**.

Tip: Um diesen Vorgang zu vereinfachen, können Sie alle Projekte zusammen mit der Anwendung in einer einzelnen Projektgruppe verwalten. Diese kann dann in der Projektverwaltung mit **Projekt**▸**Alle Projekte compilieren** compiliert werden.

Siehe auch

Sprachen zu einem Projekt hinzufügen (siehe Seite 86)

Ressourcendateien im Translation-Manager bearbeiten (siehe Seite 88)

Externen Translation-Manager einrichten (siehe Seite 91)

1.77 Externen Translation-Manager verwenden

Wenn die RAD Studio-IDE nicht zur Verfügung steht, kann anstelle des internen der externe Translation-Manager (ETM) zur Lokalisierung verwendet werden. Der ETM wird ähnlich verwendet wie der interne Translation-Manager.

Anmerkung: Bevor Sie mit der Arbeit beginnen, müssen Sie den ETM auf Ihrem Computer installieren und einrichten. Weitere Informationen erhalten Sie über den Link am Ende dieses Themas.

So starten Sie den ETM:

1. Um den ETM von der Befehlszeile aus zu starten, geben Sie folgenden Befehl ein: `etm.exe [Dateien]` Hierbei steht `[Dateien]` für die optionale Projektgruppdatei bzw. die Projektdateien.
2. Um den ETM aus dem Windows-Explorer heraus zu starten, doppelklicken Sie auf `etm.exe`

So lokalisieren Sie eine Anwendung mit dem ETM:

1. Wählen Sie im ETM **Datei** **Öffnen**, und öffnen Sie das zu übersetzende Projekt.
2. Klicken Sie auf die Registerkarte **Arbeitsbereich**.
3. Erweitern Sie die Baumansicht des Projekts, sodass die zu bearbeitenden Ressourcendateien angezeigt werden.
- Tip:** Verwenden Sie hierzu die Symbole zum Ein- und Ausblenden in der Symbolleiste über der Baumansicht.
4. Klicken Sie auf die Unit-Datei, die Sie bearbeiten möchten. Die in der Datei enthaltenen Ressourcen-Strings werden in einer Tabelle im rechten Bereich angezeigt.
5. Klicken Sie auf das Feld, das Sie bearbeiten möchten, und führen Sie dann einen der folgenden Schritte aus:
 - Geben Sie den neuen Text direkt in die Tabelle ein.
 - Klicken Sie mit der rechten Maustaste auf das Feld, und wählen Sie **Bearbeiten**, um den String in einem Dialogfeld zu bearbeiten.
 - Klicken Sie in der Symbolleiste über der Tabelle auf das Symbol **Mehrzeilen-Editor**
6. Sie können bei Bedarf einen Kommentar in das Feld **Kommentar** eingeben.
7. Optional können Sie über die Dropdown-Liste im Feld **Status** den Übersetzungsstatus des Strings festlegen.
8. Klicken Sie in der Symbolleiste über der Tabelle auf das Symbol **Übersetzung speichern**, um die Ressourcendatei zu aktualisieren.

Nachdem Sie die Übersetzungen fertig gestellt haben, können Sie die übersetzten Dateien wieder dem Entwickler geben, damit er sie dem Projekt hinzufügt.

So entfernen Sie Sprachen aus einem Projekt:

1. Öffnen Sie ein Projekt.
2. Deaktivieren Sie auf der Registerkarte **Sprachen** das Kontrollkästchen für die Sprache, die Sie entfernen möchten.
3. Aktivieren Sie die Registerkarte **Dateien**, und klicken Sie auf die Schaltfläche **Aktualisierung** ausführen.

Der ETM entfernt die ausgewählten Assemblierungen oder DLLs aus dem Projekt, löscht aber weder die Assemblierungen/DLLs noch ihren Quelltext oder die Verzeichnisse, in denen sie sich befinden.

Siehe auch

Externen Translation-Manager einrichten (siehe Seite 91)

1.78 Konfigurieren des Speichermanagers

Dieser Abschnitt beschreibt, wie der Speichermanager konfiguriert wird.

Anmerkung: Einige Konfigurationseinstellungen des Speichermanagers können geändert werden, während der Speichermanager verwendet wird. Alle Konfigurationseinstellungen sind globale Einstellungen und wirken sich auf alle Threads aus, die den Speichermanager verwenden. Falls nicht anders angegeben, sind alle Funktionen und Prozeduren Thread-sicher.

Diese Konfigurationsoptionen gelten nur für den lokalen Speichermanager. Das Setzen dieser Optionen in einer Bibliothek hat keine Auswirkungen, wenn die Bibliothek den Speichermanager der Hauptanwendung gemeinsam nutzt.

So setzen Sie die minimale Blockausrichtung für den Speichermanager:

1. Ermitteln Sie mit der Funktion GetMinimumBlockAlignment die aktuelle, minimale Blockausrichtung.
2. Wählen Sie die geeignete Speicherblockausrichtung für Ihre Anwendung aus. Verfügbare Blockausrichtungen sind 8 Byte (mba8byte) und 16 Byte (mba16byte).
3. Zum Ändern der Speicherblockausrichtung verwenden Sie die Prozedur SetMinimumBlockAlignment.

Anmerkung: Bei Speicher, der mit dem Speichermanager zugewiesen wird, ist gewährleistet, dass er an mindestens 8-Byte-Grenzen ausgerichtet ist. Eine 16-Byte-Ausrichtung ist hilfreich, wenn Speicherblöcke mit SSE-Anweisungen manipuliert werden, könnte aber ein Speicherverwendungs-Overhead vergrößern.

So werden Sie beim Schließen über Speicherlecks informiert:

1. Setzen Sie die globale Variable ReportMemoryLeaksOnShutdown auf **True**.
2. Wenn der Speichermanager geschlossen wird, durchsucht er den Speicher-Pool und meldet alle nicht registrierten Speicherlecks in einem Meldungsdialogfeld. Um Speicherlecks zu registrieren oder deren Registrierung aufzuheben, verwenden Sie die Prozeduren RegisterExpectedMemoryLeak bzw. UnregisterExpectedMemoryLeak.

Anmerkung: Der Speichermanager kann über Speicher informieren, der zugewiesen, aber zum Zeitpunkt des Schließens des Speichermanagers noch nicht freigegeben wurde. Solche Speicherblöcke werden *Speicherlecks* genannt und sind oft das Ergebnis von Programmierfehlern. Der Vorgabewert für ReportMemoryLeaksOnShutdown ist **False**.

Die Klasse eines Lecks wird durch das erste DWord im Block bestimmt. Die gemeldeten Leckklassen sind oft nicht 100% richtig. Ein Leck wird als String-Leck gemeldet, falls es ein **AnsiString** zu sein scheint. Wenn der Speichermanager den Typ des Lecks nicht feststellen kann, wird es als Typ "unbekannte Klasse" gemeldet.

So behandeln Sie Thread-Konkurrenzen im Speichermanager:

1. Setzen Sie die globale Variable NeverSleepOnMMThreadContention auf **True**.
2. Wenn eine Thread-Konkurrenz im Speichermanager auftritt, wartet er in einer Schleife, bis die Konkurrenz aufgelöst ist.

Anmerkung: Der Speichermanager ist eine gemeinsam genutzte Ressource. Wenn viele Threads in der Anwendung gleichzeitig eine Speichermanageroperation auszuführen versuchen, muss ein Thread (oder mehrere) evtl. warten, bis eine anstehende Operation eines anderen Threads abgeschlossen ist, bevor der Thread fortgesetzt werden kann. Diese Situation wird *Thread-Konkurrenz* genannt. Wenn eine Thread-Konkurrenz im Speichermanager auftritt, wird als Standardverhalten die verbleibende Zeit im Zeitabschnitt des Threads freigegeben. Wenn die Ressource beim Eintritt des Threads in den nächsten Zeitabschnitt noch nicht verfügbar ist, ruft der Speichermanager die Betriebssystemprozedur `Sleep` auf, damit vor dem nächsten Versuch länger gewartet wird (ca. 20 Millisekunden).

Dieses Verhalten funktioniert gut bei Rechnern mit Einzel- oder dualen CPUs sowie wenn das Verhältnis der Anzahl der laufenden Threads zur Anzahl der CPUs relativ hoch (größer als 2:1) ist. In anderen Situationen kann eine bessere Performance erzielt werden, indem der Thread in eine Warteschlange gestellt wird, bis die Ressource verfügbar wird. Wenn `NeverSleepOnMMThreadContention` **True** ist, tritt der Speichermanager in eine Warteschleife ein. Der Vorgabewert für `NeverSleepOnMMThreadContention` ist **False**.

Siehe auch

[Speicherverwaltung](#) (siehe Seite 1017)

[Vergrößern des Adressraums des Speichermanagers über 2 GB hinaus](#) (siehe Seite 97)

[Registrieren von Speicherlecks](#) (siehe Seite 100)

[Überwachen des Speichermanagers](#) (siehe Seite 99)

[Gemeinsame Nutzung von Speicher](#) (siehe Seite 101)

1.79 Vergrößern des Speicheradressraums

Dieser Abschnitt beschreibt, wie der Adressraum des Speichermanagers über 2 GB hinaus erweitert wird.

Anmerkung: Die Standardgröße für den Adressraum für den Benutzermodus für eine Win32-Anwendung beträgt 2 GB, er kann aber optional auf 3 GB unter 32-Bit-Windows und auf 4 GB unter 64-Bit-Windows erweitert werden. Der Adressraum ist immer ein wenig fragmentiert. Daher ist es unwahrscheinlich, dass eine GetMem-Anforderung auf einen einzelnen zusammenhängenden Block größer als 1 GB erfolgreich sein wird – auch nicht bei einem Adressraum von 4 GB.

So aktivieren und verwenden Sie einen größeren Adressraum:

1. Vergewissern Sie sich, dass das Betriebssystem einen größeren Adressraum unterstützt. Ein Adressraum für den Benutzermodus von mehr als 2 GB wird von 64-Bit-Windows-Editionen und auch von 32-Bit-Editionen unterstützt, die in der Datei `boot.ini` die Option `/3GB` gesetzt haben.
2. Setzen Sie die geeignete Linker-Direktive. Das Betriebssystem muss über ein Flag im Header der ausführbaren Datei darüber informiert werden, dass die Anwendung einen Adressraum für den Benutzermodus von mehr als 2 GB unterstützt, ansonsten werden nur 2 GB bereitgestellt. Um dieses Flag zu setzen, geben Sie in der `.dpr`-Datei der Anwendung `{$SetPEFlags IMAGE_FILE_LARGE_ADDRESS_AWARE}` ein.
3. Stellen Sie sicher, dass alle Bibliotheken und Komponenten von Fremdherstellern den größeren Adressraum unterstützen. Bei einem 2-GB-Adressraum ist das hohe Bit aller Zeiger immer 0, daher kann ein größerer Adressraum Fehler in der Zeigerarithmetik aufdecken, die zuvor keine Symptome gezeigt haben. Solche Fehler werden typischerweise verursacht, wenn Zeiger bei der Zeigerarithmetik oder bei Zeigervergleichen in Integertypen anstatt in Kardinaltypen umgewandelt werden.

Anmerkung: Bei Speicher, der mit dem Speichermanager zugewiesen wird, ist gewährleistet, dass er an mindestens 8-Byte-Grenzen ausgerichtet ist. Eine 16-Byte-Ausrichtung ist hilfreich, wenn Speicherblöcke mit SSE-Anweisungen manipuliert werden, könnte aber ein Speicherwendungs-Overhead vergrößern. Die gesetzte minimale Blockausrichtung für zukünftige Zuweisungen kann mit `SetMinimumBlockAlignment` gesetzt werden.

Siehe auch

[Speicherverwaltung](#) (siehe Seite 1017)

[Konfigurieren des Speichermanagers](#) (siehe Seite 95)

[Registrieren von Speicherlecks](#) (siehe Seite 100)

[Überwachen des Speichermanagers](#) (siehe Seite 99)

[Gemeinsame Nutzung von Speicher](#) (siehe Seite 101)

1.80 Speicherverwaltung

Dieser Abschnitt enthält Informationen über die Verwendung des Speichermanagers. Folgende Themen werden behandelt: Konfigurieren des Speichermanagers, Vergrößern des Speicheradressraums, Überwachen des Speichermanagers, Verwenden des Speicherabilds, gemeinsames Nutzen von Speicher und Verwalten von Speicherlecks.

1.81 Überwachen der Speicherverwendung

Dieser Abschnitt beschreibt, wie der Status des Speichermanagers überwacht wird.

Der Speichermanager stellt zwei Prozeduren bereit, mit denen die Anwendung ihre eigene Speicherverwendung und den Status des Prozessadressraums überwachen kann. Beide Funktionen sind Thread-sicher.

So überwachen Sie die Speicherverwendung Ihrer Anwendung:

1. Rufen Sie die Prozedur GetMemoryManagerState auf.
2. Untersuchen Sie die Struktur TMemoryManagerState und entnehmen Sie ihr die benötigten Statusinformationen des Speichermanagers. Die Struktur enthält Felder, die die Gesamtanzahl der Zuweisungen, die Summe ihrer Größen und den gesamten reservierten Adressraum aufführen. Die Statistik ist in drei Kategorien unterteilt: kleine, mittlere und große Zuweisungen.

So ermitteln Sie ein Abbild des Speicheradressraums für einen Prozess:

1. Rufen Sie die Prozedur GetMemoryMap auf.
2. Untersuchen Sie das Array TMemoryMap und entnehmen Sie ihm die benötigten Informationen über den Adressraum des Prozesses. Das Array enthält für jeden möglichen 64-K-Block im Adressraum des Prozesses einen TChunkStatus-Eintrag.

Siehe auch

Speicherverwaltung ([siehe Seite 1017](#))

Konfigurieren des Speichermanagers ([siehe Seite 95](#))

Vergrößern des Adressraums des Speichermanagers über 2 GB hinaus ([siehe Seite 97](#))

Registrieren von Speicherlecks ([siehe Seite 100](#))

Gemeinsame Nutzung von Speicher ([siehe Seite 101](#))

1.82 Registrieren von Speicherlecks

Dieser Abschnitt beschreibt, wie erwartete Speicherlecks registriert und die Registrierung wieder aufgehoben wird.

Wenn Sie Speicher zuweisen, der wahrscheinlich nicht freigegeben wird, können Sie ihn beim Speichermanager registrieren. Der Speichermanager fügt ihn zu einer Liste mit Bereichen hinzu, die bei der Prüfung auf Speicherlecks ignoriert werden sollen. Wenn Sie die Registrierung einer Speicherposition aufheben, entfernt der Speichermanager sie aus der Liste mit den erwarteten Speicherlecks.

So registrieren Sie ein erwartetes Speicherleck:

1. Identifizieren Sie den Zeiger auf den Speicherbereich, der wahrscheinlich nicht freigegeben wird.
2. Übergeben Sie den Zeiger an RegisterExpectedMemoryLeak.

So heben Sie die Registrierung eines erwarteten Speicherlecks auf:

1. Identifizieren Sie den Zeiger auf den Speicherbereich, dessen Registrierung Sie aufheben möchten.
2. Übergeben Sie den Zeiger an UnregisterExpectedMemoryLeak.

Siehe auch

Speicherverwaltung ([siehe Seite 1017](#))

Konfigurieren des Speichermanagers ([siehe Seite 95](#))

Vergrößern des Adressraums des Speichermanagers über 2 GB hinaus ([siehe Seite 97](#))

Überwachen des Speichermanagers ([siehe Seite 99](#))

Gemeinsame Nutzung von Speicher ([siehe Seite 101](#))

1.83 Gemeinsame Nutzung von Speicher

Dieser Abschnitt beschreibt, wie mit dem Speichermanager Speicher gemeinsam genutzt wird. Wenn unter Win32 eine DLL Routinen exportiert, die lange Strings oder dynamische Arrays als Parameter oder als Funktionsergebnis übergeben (entweder direkt oder in Records bzw. Objekten), müssen die DLL und ihre Client-Anwendungen (oder DLLs) denselben Speichermanager verwenden. Dasselbe gilt, wenn eine Anwendung oder DLL mit **New** oder **GetMem** Speicherplatz reserviert, der in einem anderen Modul durch einen Aufruf von **Dispose** oder **FreeMem** wieder freigegeben wird. Es gibt zwei sich gegenseitig ausschließende Methoden, über die der Speichermanager von einer Anwendung und ihren Bibliotheken gemeinsam genutzt werden kann: **ShareMem** und **SimpleShareMem**.

Anmerkung: Wenn eine DLL statisch an eine Anwendung gelinkt ist, wird die DLL vor der Anwendung initialisiert. Die Anwendung verwendet den Speichermanager der DLL, wenn die Methode **SimpleShareMem** für beide aufgerufen wird. Nur das Modul, das seinen Speichermanager bereitstellt, kann die Einstellungen des Speichermanagers ändern und die Statistik des Speichermanagers abrufen. Das Ändern der Einstellungen in anderen Modulen hat keine Auswirkungen, da ihre Speichermanager nicht verwendet werden.

Es ist möglich, aber selten erforderlich, den Mechanismus des Speichermanagers zur gemeinsamen Nutzung von Speicher manuell zu steuern.

So verwenden Sie ShareMem:

1. Geben Sie **ShareMem** als erste Unit in der **uses**-Klausel des Programms und der Bibliothek an. Ihre Module werden abhängig von der externen Bibliothek *BORLNDMM.DLL*, wodurch sie zugewiesenen Speicher dynamisch gemeinsam nutzen können.
2. Verteilen Sie *BORLNDMM.DLL* mit Ihrer Anwendung oder DLL, die **ShareMem** verwendet. Wenn eine Anwendung oder DLL **ShareMem** verwendet, ersetzt *BORLNDMM.DLL* den Speichermanager dieser Anwendung oder DLL.

So verwenden Sie SimpleShareMem:

1. Geben Sie **SimpleShareMem** als erste Unit in der **uses**-Klausel des Programms und der Bibliothek aller Module an. Das Modul, das als erstes initialisiert wird, stellt seinen Speichermanager für die gemeinsame Nutzung zur Verfügung. Alle Module, die danach initialisiert werden, verwenden den Speichermanager des ersten Moduls.
2. Das Modul, das als erstes initialisiert wird, stellt seinen Speichermanager für die gemeinsame Nutzung zur Verfügung. Alle Module, die danach initialisiert werden, verwenden den Speichermanager des ersten Moduls.

Siehe auch

Speicherverwaltung (siehe Seite 1017)

Konfigurieren des Speichermanagers (siehe Seite 95)

Registrieren von Speicherlecks (siehe Seite 100)

Vergrößern des Adressraums des Speichermanagers über 2 GB hinaus (siehe Seite 97)

Überwachen des Speichermanagers (siehe Seite 99)

1.84 Together konfigurieren

Together bietet weit gehende Konfigurationsmöglichkeiten. Im Dialogfeld Optionen können Sie die Modellierungsfunktionen an Ihre Anforderungen anpassen.

Das Dialogfeld Optionen enthält eine Reihe von Einstellungen für die Anpassung von Diagrammen. Sie können das Erscheinungsbild und das Layout der Diagramme konfigurieren und Schrifteigenschaften, das Format für Elemente und die Detailebene festlegen.

So passen Sie die Together-Einstellungen an:

1. Wählen Sie im Hauptmenü **Tools**▶**Optionen**.
2. Erweitern Sie im Dialogfeld Optionen die Kategorie Together.
3. Wählen Sie die gewünschte Optionsebene aus.
4. Für Optionen auf Projekt- und Diagrammebene wählen Sie das Projekt oder Diagramm, auf das die Konfiguration angewendet werden soll. Klicken Sie dazu auf die Auswahlshaltflächen der entsprechenden Felder, und wählen Sie das gewünschte Projekt oder Diagramm im Modell aus.
5. Klicken Sie auf die gewünschte Unterkategorie.
6. Bearbeiten Sie die Konfigurationsoptionen nach Bedarf.
7. Klicken Sie auf OK, um die Änderungen zu übernehmen und das Dialogfeld zu schließen.

Sie können Konfigurationsoptionen auf einer übergeordneten Ebene einstellen und alle Änderungen auf den untergeordneten Ebenen deaktivieren:

So deaktivieren Sie Konfigurationsänderungen:

1. Wählen Sie im Hauptmenü **Tools**▶**Optionen**.
2. Klicken Sie auf die Kategorie Together, um sie zu erweitern.
3. Wählen Sie die benötigte Unterkategorie aus (Standard, Projektgruppe oder Projekt).
4. Aktivieren Sie die Option Unterebenen deaktivieren.

Siehe auch

Optionsebenen (siehe Seite 1280)

Optionswert-Editoren (siehe Seite 1280)

1.85 Refactoring: Parameter ändern

So ändern Sie Parameter:

1. Wählen Sie eine Methode in der Diagrammansicht, der Modellansicht oder im Editor aus.
2. Wählen Sie im Hauptmenü Refactoring>Parameter ändern.

Tip: Alternativ können Sie mit der rechten Maustaste klicken und im Kontextmenü Refactoring>Parameter ändern wählen.

3. Markieren Sie im angezeigten Dialogfeld den Parameter in der Liste, und wählen Sie die gewünschte Aktion:
 - Klicken Sie auf Hinzufügen, um einen neuen Parameter hinzuzufügen, und legen Sie den Namen, den Typ und den Standardwert für den Parameter fest.
 - Klicken Sie auf Entfernen, um den Parameter zu löschen.
 - Wenn Sie den Parameter umbenennen möchten, klicken Sie auf das Feld Name und bearbeiten den Parameternamen mit dem internen Editor.
4. Aktivieren Sie gegebenenfalls die Option Refactoring für Vorfahren.
5. Verwenden Sie die Option Vorschau der Verwendung nach Bedarf.
 - Wenn Sie diese Option aktivieren und dann auf **OK** klicken, wird das Fenster Refactoring geöffnet, und Sie können die Refactoring-Operation vor der Übernahme überprüfen. Klicken Sie auf die Schaltfläche Refactoring ausführen, um die Änderungen zu bestätigen. Nach der Durchführung des Refactoring können Sie bei Bedarf die Befehle Rückgängig und Wiederherstellen verwenden.
 - Wenn Sie bei deaktiverter Option auf **OK** klicken, wird das Fenster Refactoring mit den abgeschlossenen Änderungen geöffnet. Nach der Durchführung des Refactoring können Sie bei Bedarf die Befehle Rückgängig und Wiederherstellen verwenden.

Siehe auch

Überblick zum Refactoring (siehe Seite 1456)

Parameter ändern (Dialogfeld) (siehe Seite 684)

1.86 Refactoring: Interfaces extrahieren

Beim Extrahieren von Interfaces ist Folgendes zu beachten:

- Es können nur nicht-statische Methoden extrahiert werden.
- Alle Methoden im extrahierten Interface sind als **public** deklariert.
- Ist der Name des neuen Interface identisch mit dem eines vorhandenen Interface im selben Namespace, werden alle Methoden in das vorhandene Interface extrahiert.

So extrahieren Sie ein Interface:

1. Wählen Sie in der Diagrammansicht oder der Modellansicht ein oder mehrere Quelltextelemente aus (Klasse, Interface, Feld, Methode, Ereignis, Eigenschaft oder Indizierer).

2. Wählen Sie im Hauptmenü **Refactoring** ▶ **Superklasse extrahieren**.

Tip: Alternativ können Sie mit der rechten Maustaste klicken und im Kontextmenü **Refactoring** ▶ **Superklasse extrahieren** wählen.

3. Geben Sie im Dialogfeld Interface extrahieren den Namen für das Interface und gegebenenfalls den Namespace ein.

4. Legen Sie durch Aktivieren bzw. Deaktivieren der entsprechenden Kontrollkästchen die Elemente fest, die in der resultierenden Superklasse oder dem Interface verwendet werden sollen.

5. Klicken Sie auf OK. Das Fenster Refactoring wird geöffnet. Hier können Sie die Refactoring-Operation vor der Übernahme überprüfen.

6. Klicken Sie auf die Schaltfläche Refactoring ausführen, um das Extrahieren abzuschließen.

Siehe auch

Überblick zum Refactoring (siehe Seite 1456)

Interface extrahieren/Superklasse extrahieren (Dialogfelder) (siehe Seite 688)

1.87 Refactoring: Methoden extrahieren

So extrahieren Sie eine Methode:

1. Öffnen Sie im Editor die Klasse oder das Interface mit dem zu extrahierenden Quelltextfragment.
2. Stellen Sie den Cursor in das gewünschte Quelltextfragment. Die Refactoring-Funktion bestimmt den Anfang und das Ende der Anweisung automatisch.
3. Wählen Sie im Hauptmenü **Refactoring** ▶ **Methode extrahieren**.

Tip: Alternativ können Sie mit der rechten Maustaste auf das Quelltextfragment klicken und im Kontextmenü **Refactoring** ▶ **Methode extrahieren** wählen.

4. Geben Sie im angezeigten Dialogfeld folgende Informationen ein:

- Den Namen der neuen Methode
- Die Sichtbarkeit (**public**, **protected**, **private**, **internal**, **internal protected**)
- Den einleitenden Kommentar
- Die Angabe, ob es sich um eine statische Methode handelt

5. Klicken Sie auf OK, um das Extrahieren abzuschließen und die neue Methode zu erstellen.

Tip:

- Bei der Anwendung des Befehls **Methode extrahieren** werden Parameter und lokale Variablen des ausgewählten Quelltextfragments als Parameter der neuen Methode übernommen.
- Das Quelltextfragment kann keine **return**-Anweisung der Originalmethode enthalten. Wenn Sie eine derartige Anweisung in das Quelltextfragment aufnehmen, wird eine Fehlermeldung angezeigt.
- Das Fragment kann nur eine einzige lokale Variable ändern. Wenn Sie diese Einschränkung nicht beachten, wird eine Fehlermeldung angezeigt.
- Ist das ausgewählte Quelltextfragment mehrfach vorhanden, müssen Sie die Fragmente an den betreffenden Stellen durch passende Methodenaufrufe ersetzen.

Siehe auch

Überblick zum Refactoring (siehe Seite 1456)

1.88 Refactoring: Superklasse extrahieren

So wenden Sie die Operation "Superklasse extrahieren" an:

1. Wählen Sie in der Diagrammansicht oder der Modellansicht ein oder mehrere Quelltextelemente aus (Klasse, Interface, Feld, Methode, Ereignis, Eigenschaft oder Indizierer).
2. Wählen Sie im Hauptmenü **Refactoring** ▶ **Superklasse extrahieren**.

Tip: Alternativ können Sie mit der rechten Maustaste klicken und im Kontextmenü **Refactoring** ▶ **Superklasse extrahieren** wählen.

3. Geben Sie im Dialogfeld Superklasse extrahieren den Namen für das Interface und gegebenenfalls den Namespace ein.
4. Legen Sie durch Aktivieren bzw. Deaktivieren der entsprechenden Kontrollkästchen die Elemente fest, die in der resultierenden Superklasse oder dem Interface verwendet werden sollen. Legen Sie gegebenenfalls fest, dass in der extrahierten Superklasse eine abstrakte Methode vorhanden ist.
5. Klicken Sie auf OK. Das Fenster Refactoring wird geöffnet. Hier können Sie die Refactoring-Operation vor der Übernahme überprüfen.
6. Klicken Sie auf die Schaltfläche Refactoring ausführen, um das Extrahieren abzuschließen.

Siehe auch

Überblick zum Refactoring (☞ siehe Seite 1456)

Interface extrahieren/Superklasse extrahieren (Dialogfelder) (☞ siehe Seite 688)

1.89 Refactoring: Inline-Variablen erstellen

So erstellen Sie eine Inline-Variable:

1. Wählen Sie eine lokale Variable im Editor aus.
2. Wählen Sie im Hauptmenü **Refactoring**▶**Inline für Variable**.

Tip: Alternativ können Sie mit der rechten Maustaste klicken und im Kontextmenü **Refactoring**▶**Inline für Variable** wählen.

Daraufhin wird in einem Dialogfeld die Anzahl der Fundstellen der Variable angezeigt, auf die der Befehl Inline für Variable angewendet wird.

3. Klicken Sie auf OK, um das Refactoring abzuschließen.

Warnung: Die Variable, aus der die Inline-Variable erzeugt wird, darf später im Quelltext nicht geändert werden. Bei einer Änderung wird folgende Fehlermeldung angezeigt: "Auf Variable wird zum Schreiben nicht zugegriffen".

Angenommen, Sie wenden den Refactoring-Befehl Inline für Variable folgendermaßen auf die lokale Variable `index` an:

```
public void findIndex() {  
    int index = 2;  
    System.Console.WriteLine("Index is: {0}", index);  
}
```

Das Ergebnis dieser Refactoring-Operation sieht folgendermaßen aus:

```
public void findIndex() {  
    System.Console.WriteLine("Index is: {0}", 2);  
}
```

Siehe auch

Überblick zum Refactoring (☞ siehe Seite 1456)

Inline für Variable (Dialogfeld) (☞ siehe Seite 691)

1.90 Refactoring: Felder einführen

So führen Sie ein Feld ein:

1. Wählen Sie einen Ausdruck im Editor aus.
2. Wählen Sie im Hauptmenü **Refactoring** ▶ **Feld einführen**.

Tip: Alternativ können Sie mit der rechten Maustaste klicken und im Kontextmenü **Refactoring** ▶ **Feld einführen** wählen.

3. Geben Sie im angezeigten Dialogfeld folgende Informationen ein:
 - Name: Geben Sie den Namen des neuen Feldes ein.
 - Sichtbarkeit: Wählen Sie im Listenfeld ein Sichtbarkeitsattribut für das neue Feld: *public*, *protected*, *private*, *internal* oder *internal protected*.
 - Initialisieren: Geben Sie an, wo das neue Feld initialisiert werden soll. Im Listenfeld stehen folgende Auswahlmöglichkeiten zur Verfügung: *Aktuelle Methode*, *Klassenkonstruktor(en)* oder *Felddeklaration*.
4. Aktivieren Sie bei Bedarf die Kontrollkästchen **Statisch** und **Alle Vorkommen ersetzen**.
5. Klicken Sie auf **OK**, um das Refactoring abzuschließen.

Siehe auch

Überblick zum Refactoring (siehe Seite 1456)

Feld einführen (Dialogfeld) (siehe Seite 692)

1.91 Refactoring: Variablen einführen

So führen Sie eine neue Variable ein:

1. Wählen Sie eine Variable im Editor aus.
2. Wählen Sie im Hauptmenü **Refactoring**▶**Variable einführen**.

Tip: Alternativ können Sie mit der rechten Maustaste klicken und im Kontextmenü **Refactoring**▶**Variable einführen** wählen.

3. Legen Sie im angezeigten Dialogfeld den Namen der neuen Variable fest. Die neue Variable hat denselben Typ wie die Originalvariable.
4. Aktivieren Sie gegebenenfalls das Kontrollkästchen Alle Vorkommen ersetzen. Im Dialogfeld Variable einführen wird die Anzahl der Fundstellen angezeigt, die durch die neue Variable ersetzt werden.

Anmerkung: Fundstellen im Quelltext vor der Position, an der die Refactoring-Operation initiiert wird, werden nicht ersetzt.

Siehe auch

- Überblick zum Refactoring (🔗 siehe Seite 1456)
Variable einführen (Dialogfeld) (🔗 siehe Seite 693)

1.92 Refactoring: Member verlagern

So verlagern Sie statische Member in eine andere Klasse:

1. Wählen Sie in der Diagrammansicht oder der Modellansicht ein oder mehrere statische Member aus.
2. Klicken Sie im Hauptmenü auf **Refactoring**▶**Verlagern**.

Tip: Sie können auch mit der rechten Maustaste auf die Auswahl klicken und im Kontextmenü **Refactoring**▶**Member verlagern** wählen.

3. Wählen Sie im Dialogfeld Member verlagern im Bereich Member verlagern die statischen Member aus, die verlagert werden sollen. Aktivieren Sie dazu die Kontrollkästchen neben den Namen der gewünschten Member.
4. Geben Sie in das Feld Zu den vollqualifizierten Namen für die Zielklasse ein, in die die ausgewählten Quelltextelemente aufgenommen werden sollen.
5. Klicken Sie auf OK.

Siehe auch

Überblick zum Refactoring (☞ siehe Seite 1456)

Member verlagern (Dialogfeld) (☞ siehe Seite 695)

1.93 Refactoring: Member in die übergeordnete oder abgeleitete Klasse verschieben

Ein Member kann entweder an die Zielposition verschoben oder dort neu erstellt werden. Im ersten Fall wird das Member an der Ausgangsposition gelöscht, im zweiten Fall bleibt es dort erhalten.

So verschieben Sie ein Member:

1. Wählen Sie ein Member in der Diagrammansicht oder der Modellansicht aus.

Tip: Stellen Sie im Editor den Cursor in den Namen des Member.

2. Wählen Sie im Haupt- oder Kontextmenü **Refactoring** > **Member in übergeordnete Klasse verschieben/Member in abgeleitete Klasse verschieben**.

3. Geben Sie im angezeigten Dialogfeld weitere Informationen ein, die für die Operation erforderlich sind.

- Markieren Sie im oberen Bereich des Dialogfelds die zu verschiebenden Member.
- Wählen Sie in der Klassenhierarchie im unteren Bereich des Dialogfelds die Zielklasse aus.

4. Klicken Sie auf OK.

5. Das Fenster Refactoring wird geöffnet. Hier können Sie die Refactoring-Operation vor der Übernahme überprüfen. Klicken Sie auf die Schaltfläche Refactoring ausführen, um die Operation abzuschließen.

Tip: Das Verschieben von Membern ist komplizierter als das Verschieben von Klassen zwischen Namespaces, da Klassen-Member sich oft gegenseitig referenzieren. Wenn bei Anwendung des Befehls Member in übergeordnete Klasse verschieben oder Member in abgeleitete Klasse verschieben die Syntaxregeln verletzt werden, da das zu verlagernde Member Referenzen auf andere Klassen-Member enthält, wird eine Warnmeldung angezeigt. Sie können dann die Verschiebung trotzdem durchführen und den resultierenden Quelltext manuell korrigieren.

Siehe auch

Überblick zum Refactoring (siehe Seite 1456)

1.94 Refactoring: Sicheres Löschen

So wenden Sie die Operation "Sicheres Löschen" auf ein Element an:

1. Wählen Sie das zu löschen Element aus.
2. Wählen Sie im Hauptmenü **Refactoring** ▶ **Sicheres Löschen**.

Tip: Sie können auch mit der rechten Maustaste auf das Element klicken und im Kontextmenü **Refactoring** ▶ **Sicheres Löschen** wählen.

3. Das Dialogfeld Sicheres Löschen wird geöffnet. Hier werden das zu löschen Element und alle seine Vorkommen (sofern vorhanden) angezeigt. Sie haben nun folgende Möglichkeiten:
 - Drücken Sie die Taste ENTF, wenn keine Verwendung des Elements gefunden wurde.
 - Klicken Sie auf Verwendung anzeigen, wenn Vorkommen des Elements gefunden wurden. Das Fenster **Refactoring** wird geöffnet. Hier können Sie die Refactoring-Operation vor der Übernahme überprüfen. Klicken Sie auf die Schaltfläche Refactoring ausführen, um das Element zu löschen.

Siehe auch

Überblick zum Refactoring (☞ siehe Seite 1456)

Sicheres Löschen (Dialogfeld) (☞ siehe Seite 705)

1.95 Together – Anleitungen

Dieser Abschnitt enthält Anleitungen zur Verwendung der Together-Features.

1.96 Die Online-Hilfe verwenden

Führen Sie einen der folgenden Schritte aus, wenn Sie bei Ihrer Arbeit Unterstützung benötigen:

1. Wenn Sie Informationen zur Funktionalität eines Bildschirmelements in einem Dialogfeld benötigen, drücken Sie die Taste F1 oder klicken auf die Schaltfläche Hilfe.
2. Zur Anzeige des Hilfethemas für einen Fensterbereich, eine Ansicht, eine Tool-Paletten-Schaltfläche oder ein anderes Element drücken Sie die Taste F1.
3. Um das Inhaltsverzeichnis der Online-Hilfe zu öffnen, wählen Sie im Hauptmenü **Hilfe**►**CodeGear-Hilfe**. Dadurch wird die Registerkarte Inhalt aktiviert.
4. Wenn Sie nach bestimmten Themen und Begriffen suchen, verwenden Sie die Registerkarte Index.
5. Wenn Sie Fragen zu RAD Studio haben, besuchen Sie den technischen Support von CodeGear unter der Adresse <http://support.borland.com>.

So filtern Sie die Hilfeinformationen:

1. Um nicht benötigte Bücher und Themen im Inhaltsverzeichnis und Index der Hilfe herauszufiltern, klicken Sie im Listenfeld **Filter** auf den entsprechenden Eintrag:
 - RAD Studio für .NET
 - RAD Studio für Win32
 - usw.
2. Wenn ein Thema relevante Informationen für verschiedene RAD Studio-Funktionssätze enthält, können Sie die entsprechende Filterung über die Schaltfläche **Filter** vornehmen.

Siehe auch

Hilfe zur Hilfe (siehe Seite 1378)

Tastenzuordnungen (siehe Seite 1335)

Together Tastaturkürzel (siehe Seite 1278)

1.97 Diagramme mit Hinweisen versehen

Verwenden Sie eines der folgenden Verfahren, um Hinweise in Diagramme einzufügen:

1. Erzeugen eines Hinweises
2. Erzeugen einer Hinweisbeziehung
3. Eingeben von Kommentaren

So erzeugen Sie einen Hinweis:

1. In der Diagrammansicht können Sie folgende Aktionen ausführen:
 - Den Hinweis durch einen Hyperlink mit einem anderen Diagramm oder Element verknüpfen.
 - Den Text bearbeiten, wenn der zugehörige interne Editor aktiv ist.
 - Die Eigenschaften eines Hinweises im Objektinspektor bearbeiten.
 - Einen Hinweis aus einem Diagramm mit Hilfe einer Verknüpfung in ein anderes Diagramm einfügen. Wählen Sie dazu im Kontextmenü eines Diagramms den Befehl **Hinzufügen > Verknüpfungen**. Im Objektinspektor des Hinweises können Sie folgende Aktionen ausführen:
 - Den Text bearbeiten.
 - Die Vorder- und Hintergrundfarbe ändern.
 - Die Nur-Text-Eigenschaft ändern.

So erzeugen Sie eine Hinweisbeziehung:

1. Klicken Sie in der Tool-Palette auf die Schaltfläche **Hinweisbeziehung**.
2. Klicken Sie in der Diagrammansicht auf das Quellelement.
3. Ziehen Sie die Beziehung mit der Maus zum Zielelement.
4. Lassen Sie die Maustaste los, sobald das zweite Element optisch hervorgehoben wird.

Tip: Mit Hilfe des Objektinspektors können Sie beide Seiten der Beziehung, Client und Anbieter, anzeigen.

So geben Sie Kommentare ein:

1. Für die Eingabe von Kommentaren in den Quelltext stehen im Objektinspektor der Klasse die Kommentar-Felder (*Autor*, *Seit*, *Version*) zur Verfügung.
2. Mit Hilfe des Editors können Sie Kommentare auch direkt in den Quelltext eingeben.

Siehe auch

Überblick zu Hinweisen in Diagrammen (siehe Seite 1448)

Ein Element erstellen (siehe Seite 142)

Verknüpfungen erstellen (siehe Seite 139)

1.98 Diagramme erstellen

Wenn Sie ein neues Diagramm erstellen, wird in der Diagrammansicht ein leerer Hintergrund angezeigt. Sie können die verschiedenen Modellelemente auf dem Hintergrund platzieren und gemäß den Anforderungen Ihres Modells Beziehungen zwischen ihnen zeichnen.

So erstellen Sie ein Diagramm:

1. Klicken Sie in der Modellansicht mit der rechten Maustaste auf das Zielprojekt.

Tip: Sie können stattdessen auch die Tastenkombination STRG+UMSCHALT+D drücken.

2. Wählen Sie den Ziel-Namespace (bzw. das Paket) in der Diagrammansicht oder der Modellansicht aus. Wenn Sie keinen benutzerdefinierten Namespace (Paket) auswählen, fügt Together dem Standarddiagramm ein neues Diagramm hinzu.

3. Wählen Sie im Kontextmenü **Hinzufügen**▶**Anderes Diagramm**.

4. Aktivieren Sie im Dialogfeld Neues Diagramm hinzufügen die Registerkarte Diagramme.

5. Wählen Sie den Diagrammtyp aus.

6. Geben Sie in das Feld Name einen Namen für das neue Diagramm ein.

7. Klicken Sie auf OK.

Ergebnis: Das neue Diagramm wird im Editorfenster auf einer neuen Registerkarte geöffnet. Sie können die Diagrammeigenschaften im Objektinspektor anzeigen und bearbeiten.

Ein Diagramm kann auch mit dem Befehl **Hyperlink**▶**Zu neuem Diagramm** im Kontextmenü der Modellansicht oder Diagrammansicht erstellt werden.

Ein neues logisches Klassendiagramm lässt sich mit Hilfe des Kontextmenüs des Projektstammknotens oder eines Namespace-Elements in der Modellansicht erzeugen. Wählen Sie entweder **Hinzufügen**▶**Klassendiagramm** oder **Hinzufügen**▶**Anderes Diagramm**. Wenn Sie den letzteren Befehl wählen, wird das Dialogfeld Neues Diagramm hinzufügen geöffnet. Wenn Sie eine Klasse, ein Interface oder einen Namespace in ein logisches Klassendiagramm einfügen, erzeugt Together den entsprechenden Quelltext oder den abgeleiteten Namespace in dem Namespace, in dem sich dieses Klassendiagramm befindet.

Siehe auch

Projekte erstellen (siehe Seite 249)

1.99 Notation für Diagramme ändern

Es gibt verschiedene Möglichkeiten, um die Diagrammnotation zu ändern:

1. Wählen Sie eine der beiden möglichen Darstellungsformen für Interfaces aus. Interfaces können als Rechtecke oder kleine Kreise dargestellt werden.
 2. In UML 2.0-Projekten kann für Interfaces auch die Kugelgelenknotation gewählt werden.
 3. Passen Sie Optionen für das Erscheinungsbild an, und treffen Sie auch eine Auswahl zwischen UML- und Sprachformat.
- Tip:** Die Notationsoptionen befinden sich in der Kategorie [Diagramm > Erscheinungsbild](#) der Together-Optionen.
4. Verwenden Sie das Profil **UML in Farbe**.
 5. Verwenden Sie Stereotypen.

Siehe auch

Diagramm-Layout – Überblick ([siehe Seite 1450](#))

Das Erscheinungsbild von Interfaces ändern ([siehe Seite 214](#))

Das Profil "UML in Farbe" verwenden ([siehe Seite 119](#))

Optionen für die Diagrammnotation ([siehe Seite 1282](#))

1.100 Raster und andere Optionen für das Erscheinungsbild verwenden

Sie können bei Bedarf ein Design-Raster auf dem Diagrammhintergrund ein- oder ausblenden und festlegen, dass die Elemente beim Platzieren oder Verschieben an der nächsten Gitterkoordinate einrasten. Das Raster wird im Dialogfeld mit den Optionen für das Diagrammerscheinungsbild konfiguriert.

So zeigen Sie das Raster an:

1. Öffnen Sie das Dialogfeld **Optionen**.
2. Wählen Sie die Kategorie **Together**▸**Diagramm**▸**Erscheinungsbild**, und öffnen Sie die Gruppe **Raster**.
3. Passen Sie Optionen an.

Anmerkung: In der Standardeinstellung wird das Raster angezeigt und die Elemente rasten ein.

Siehe auch

Optionen für das Erscheinungsbild von Diagrammen (siehe Seite 1282)

1.101 Das Profil "UML in Farbe" verwenden

So aktivieren bzw. deaktivieren Sie das Profil "UML in Farbe":

1. Öffnen Sie im Dialogfeld Optionen die Kategorie **Together**▸**(Ebene)**▸**Diagramm**▸**Erscheinungsbild**.

Tip: Sie können das Profil für die Projektgruppe, das Projekt oder die Diagrammebene aktivieren bzw. deaktivieren.

2. Setzen Sie die Option UML in Farbe aktivieren auf **True**, um das Profil zu aktivieren.
3. Passen Sie bei Bedarf die im Profil verwendeten Farben an.
4. Schließen Sie das Dialogfeld Optionen.

So zeichnen Sie UML-Knoten in Farbe:

1. Wählen oder erstellen Sie einen Klassifizierer.
2. Öffnen Sie den Objektinspektor.
3. Weisen Sie ein Stereotyp zu, das vom Profil "UML in Farbe" unterstützt wird (beispielsweise *role*).

Ergebnis: Der Klassifizierer ändert seine Farbe gemäß den im Dialogfeld Optionen festgelegten Einstellungen.

Siehe auch

Unterstützte UML-Spezifikationen (siehe Seite 1447)

1.102 Modellelemente ausrichten

Alle oder ausgewählte Modellelemente in einem Diagramm können automatisch neu angeordnet werden.

So richten Sie Modellelemente in einem Diagramm aus:

1. Wählen Sie mehrere Knoten oder innere Klassifizierer in einem Diagramm aus.
2. Klicken Sie mit der rechten Maustaste, und wählen Sie im Kontextmenü **Ausrichtung**►(**Algorithmus**). Die folgenden Algorithmen stehen zur Verfügung:
 - Oben
 - Unten
 - Rechts
 - Links
 - X zentrieren
 - Y zentrieren

Siehe auch

Automatisches Layout für Diagramme (siehe Seite 127)

1.103 Den Typ einer Beziehung ändern

Verwenden Sie eines der folgenden Verfahren, um den Typ einer Beziehung zu ändern:

1. Legen Sie den Beziehungstyp im Objektinspektor fest.
2. Weisen Sie den Beziehungstyp mit Hilfe des Kontextmenüs zu.

So legen Sie den Beziehungstyp im Objektinspektor fest:

1. Wählen Sie **Ansicht** | Objektinspektor, wenn der Objektinspektor noch nicht geöffnet ist.
2. Wählen Sie eine Beziehung im Diagramm aus. Die Eigenschaften für die Beziehung werden im Objektinspektor angezeigt.
3. Wählen Sie im Objektinspektor das Feld **Typ** aus.
4. Klicken Sie auf den Pfeil der Dropdown-Liste, und wählen Sie die entsprechende Eigenschaft in der Liste aus. Folgende Auswahlmöglichkeiten stehen zur Verfügung: **Assoziation**, **Aggregation** und **Komposition**.

So weisen Sie den Beziehungstyp mit Hilfe des Kontextmenüs zu:

1. Klicken Sie im Diagramm mit der rechten Maustaste auf eine Beziehung.
2. Klicken Sie im Kontextmenü auf Beziehungstyp.

Siehe auch

Eine einfache Beziehung erstellen (siehe Seite 141)

Klassendiagrammbeziehungen (siehe Seite 1296)

1.104 Diagramme schließen

So schließen Sie ein Diagramm:

1. Wechseln Sie in die Diagrammansicht.
2. Klicken Sie auf das Kreuzsymbol, um die aktuelle Ansicht zu schließen.

Anmerkung: Durch das Schließen in der Diagrammansicht wird das Diagramm nicht aus dem Projekt entfernt.

Siehe auch

Diagramme – Überblick (siehe Seite 1447)

1.105 Modellelemente kopieren und einfügen

Verschieben- und Kopieren-Operationen können per Drag&Drop, mit Kontextmenübefehlen oder mit Hilfe von Tastaturkürzeln durchgeführt werden.

Anmerkung: Sie können ein komplettes Diagramm verschieben oder kopieren. Die im Diagramm adressierten Elemente werden dabei nicht kopiert. Das neue Diagramm enthält Verknüpfungen zu diesen Elementen.

So kopieren Sie ein Element:

1. Wählen Sie das bzw. die zu kopierenden Elemente aus.
2. Führen Sie einen der folgenden Schritte aus:
 - Klicken Sie mit der rechten Maustaste, und wählen Sie im Kontextmenü Kopieren.
 - Drücken Sie die Tastenkombination **STRG+C**.
3. Führen Sie einen der folgenden Schritte aus:
 - Klicken Sie mit der rechten Maustaste auf die Zielposition, und wählen Sie im Kontextmenü den Befehl Einfügen.
 - Markieren Sie die Zielposition, und drücken Sie die Tastenkombination **STRG+V**.

Siehe auch

[Ein Element erstellen](#) (siehe Seite 142)

[Tastaturkürzel](#) (siehe Seite 1278)

1.106 Diagramme löschen

Warnung: Das automatisch erzeugte Projekt-Namespace-Diagramm (Paket-Diagramm) kann nicht gelöscht werden.

So löschen Sie ein Diagramm:

1. Wählen Sie in der Modellansicht das Diagramm aus, das gelöscht werden soll.
2. Wählen Sie im Kontextmenü den Befehl **Löschen**.
3. Bestätigen Sie die Löschung, wenn Sie dazu aufgefordert werden.

Ergebnis: Das Diagramm wird aus dem Projekt entfernt.

Siehe auch

Diagramme erstellen ( siehe Seite 116)

Diagramme schließen ( siehe Seite 122)

1.107 Hyperlinks in Diagrammen

Das Erstellen, Anzeigen, Entfernen oder Verwenden eines Hyperlinks erfolgt über den Befehl **Hyperlinks** im Kontextmenü des Diagramms.

Sie können folgende Aktionen mit Hyperlinks durchführen:

1. Erzeugen eines Hyperlinks zu einem vorhandenen Diagramm oder Element.
2. Erzeugen eines Hyperlinks zu einem neuen Diagramm.
3. Erzeugen eines Hyperlinks zu einem externen URL oder einer externen Datei.
4. Verwenden von Hyperlinks.
5. Entfernen eines Hyperlinks.

So erzeugen Sie einen Hyperlink zu einem vorhandenen Diagramm oder Element:

1. Öffnen Sie das Diagramm, in dem der Hyperlink erstellt werden soll (oder erstellen Sie ein neues Diagramm).
2. Wählen Sie das Element aus, das mit einem anderen Diagramm oder Element verknüpft werden soll.
3. Wenn Sie das gesamte Diagramm verknüpfen möchten, klicken Sie auf den Diagrammhintergrund, um die Auswahl aller Elemente aufzuheben.

Anmerkung: Wählen Sie zur Erstellung eines Hyperlinks nicht den Namespace in der Modellansicht aus. Erweitern Sie stattdessen den Namespace-Knoten, und wählen Sie das gewünschte Diagramm aus.

4. Klicken Sie mit der rechten Maustaste, und wählen Sie **Hyperlinks > Bearbeiten**. Das Dialogfeld Hyperlinks bearbeiten (Auswahlmanager) wird geöffnet.
5. Aktivieren Sie die Registerkarte **Modellelemente**, um den Bereich mit der hierarchischen Ansicht des verfügbaren Projektinhalts in der Projektmappe anzuzeigen.
6. Wählen Sie das gewünschte Diagramm oder Element in der Liste aus, und klicken Sie auf Hinzufügen.
7. Erweitern Sie für die Auswahl von Elementen die Diagrammknoten auf der Registerkarte Modellelemente.
8. Um ein Element aus der Auswahlliste zu entfernen, wählen Sie es aus und klicken auf Entfernen.
9. Klicken Sie auf OK, um das Dialogfeld zu schließen und den Link zu erstellen.

So erzeugen Sie einen Hyperlink zu einem neuen Diagramm:

1. Öffnen Sie ein Diagramm in der Diagrammansicht, oder wählen Sie es in der Modellansicht aus.
2. Wählen Sie im Kontextmenü **Hyperlinks > Zu neuem Diagramm**.
3. Wählen Sie im Dialogfeld Neues Diagramm hinzufügen den Diagrammtyp aus, geben Sie den Diagrammnamen ein, und klicken Sie auf **OK**.

So erzeugen Sie einen Hyperlink zu einem externen URL oder einer externen Datei:

1. Öffnen Sie das Diagramm, in dem der Hyperlink erstellt werden soll (oder erstellen Sie ein neues Diagramm).
2. Wählen Sie das Element aus, das mit dem externen Dokument verknüpft werden soll. Wenn Sie das gesamte Diagramm verknüpfen möchten, klicken Sie auf den Diagrammhintergrund, um die Auswahl aller Elemente aufzuheben.
3. Klicken Sie mit der rechten Maustaste, und wählen Sie **Hyperlinks > Bearbeiten**. Das Dialogfeld Hyperlinks bearbeiten wird geöffnet.
4. Aktivieren Sie die Registerkarte Externe Dokumente, um die Liste Zuletzt verwendete Dokumente anzuzeigen, in der die

zuletzt ausgewählten Dateien und URLs enthalten sind.

5. So fügen Sie eine Datei in die Liste **Zuletzt verwendete Dokumente** ein:

1. Klicken Sie auf Durchsuchen. Das Dialogfeld Datei öffnen wird angezeigt.
2. Navigieren Sie zu der gewünschten Datei, und klicken Sie auf Öffnen.

6. So fügen Sie einen URL in die Liste **Zuletzt verwendete Dokumente** ein:

3. Klicken Sie auf URL.
4. Geben Sie im angezeigten Dialogfeld den gewünschten URL ein, und klicken Sie auf OK.

Tip: Sie können einen Hyperlink zu einem externen Dokument erstellen, indem Sie einen relativen URL-Pfad eingeben.

7. Um ein Element aus der Auswahlliste zu entfernen, wählen Sie es aus und klicken auf Entfernen.

8. Wenn Sie die Liste **Zuletzt verwendete Dokumente** leeren möchten, klicken Sie auf Löschen.

Anmerkung: Elemente, die in die Liste **Zuletzt verwendete Dokumente** eingefügt werden, gehören nicht zu einem bestimmten Projekt oder einer bestimmten Projektgruppe.

9. Klicken Sie auf OK, um das Dialogfeld zu schließen und den Link zu erstellen.

So verwenden Sie Hyperlinks:

1. Um Hyperlinks zu einem Diagramm, Element oder externen Dokument anzuzeigen, klicken Sie mit der rechten Maustaste auf den Diagrammhintergrund oder das Element und wählen im Kontextmenü den Befehl **Hyperlinks**. Alle erzeugten Hyperlinks werden daraufhin im **Hyperlinks**-Untermenü angezeigt. Alle Namen von Elementen in einem Diagramm, die über Hyperlinks verknüpft sind, werden in blauer Schrift dargestellt. Wenn Sie einen Link im Untermenü auswählen, wird das entsprechende Element in der Diagrammansicht markiert.
2. Nachdem Sie Hyperlinks für ein ausgewähltes Diagramm oder Element erstellt haben, können Sie über deren Kontextmenüs zu den verknüpften Ressourcen navigieren.

Anmerkung: Wenn Sie zu einem verknüpften Diagramm navigieren, wird dieses in der Diagrammansicht geöffnet. Ist das verknüpfte Diagramm bereits geöffnet, wird es zum aktuellen Diagramm. Wenn Sie zu einem verknüpften Element navigieren, wird dessen übergeordnetes Diagramm geöffnet bzw. aktiviert. Im Diagramm wird ein Bildlauf zu dem verknüpften Element durchgeführt, und dieses wird ausgewählt.

So entfernen Sie einen Hyperlink:

1. Öffnen Sie das Diagramm, in dem sich der Link befindet, der entfernt werden soll.
2. Wählen Sie im Kontextmenü des Diagramms oder Elements **Hyperlinks** ▶ **Bearbeiten**. Das Dialogfeld **Hyperlinks bearbeiten** wird geöffnet.
3. Klicken Sie in der Auswahlliste auf der rechten Seite des Dialogfelds auf den Hyperlink, der entfernt werden soll.
4. Klicken Sie auf Entfernen.
5. Klicken Sie auf OK, um das Dialogfeld zu schließen.

Anmerkung: Wenn Sie einen Hyperlink eines bestimmten Elements entfernen möchten, müssen Sie das Element zuerst auswählen. Danach wählen Sie im Kontextmenü **Hyperlinks** ▶ **Bearbeiten**.

Siehe auch

Überblick zu Hyperlinks (siehe Seite 1450)

1.108 Automatisches Layout für Diagramme

Sie haben folgende Möglichkeiten, um das Layout eines Diagramms zu bestimmen:

1. Klicken Sie mit der rechten Maustaste auf den Diagrammhintergrund.
2. Wählen Sie im Kontextmenü zunächst den Befehl **Layout** und danach im angezeigten Untermenü einen weiteren Befehl. Das Untermenü des Befehls **Layout** enthält mehrere Layout-Befehle:
 - **Vollständiges Layout:** Das Layout aller Elemente wird entsprechend dem Layout-Algorithmus festgelegt, der für das aktuelle Diagramm definiert ist.
 - **Layout zum Drucken:** Das Layout aller Elemente wird anhand des Together-Algorithmus festgelegt, unabhängig von der auf einer anderen Ebene gewählten Option.
 - **Alle Beziehungen routen:** Die Beziehungen werden durch Entfernen von Umlenkpunkten optimiert.
 - **Größen optimieren:** Alle Elemente im Diagramm werden auf eine optimale Größe vergrößert oder verkleinert.

Anmerkung: Die Layout-Befehle **Alle Beziehungen routen** und **Größen optimieren** werden auch in den Kontextmenüs der betreffenden Diagrammelemente angeboten. Mit dem Befehl **Alle Beziehungen routen** werden die Verknüpfungen durch Entfernen von Umlenkpunkten optimiert. Mit dem Befehl **Größen optimieren** wird ein Element auf die optimale Größe verkleinert oder vergrößert, wobei genügend Platz für die Beschriftung und eventuell vorhandene Unterelemente frei bleibt.

Tip: Wenn Sie das Layout für die innere Substruktur in Diagrammen aktivieren möchten, wählen Sie im Dialogfeld **Optionen** die Option **Rekursiv (Ebene) | Diagramm | Layout | Allgemein**.

So richten Sie das Layout für Diagramme ein:

1. Wählen Sie im Hauptmenü **Tools | Optionen**.
2. Wählen Sie auf der gewünschten Ebene die Kategorie **Together| (Ebene) | Diagramm | Layout**.
3. Erweitern Sie den Knoten für den gewünschten Algorithmus.
4. Legen Sie die algorithmuspezifischen Optionen (falls vorhanden) fest, und übernehmen Sie die Änderungen.

Ergebnis: Die Ergebnisse werden sichtbar, wenn Sie einen der Layout-Befehle auf das Diagramm anwenden.

Über das Kontextmenü der Diagrammansicht kann auf die Together-Funktionen zur automatischen Layout-Optimierung zugegriffen werden.

Siehe auch

Überblick zur Modellierung (siehe Seite 1446)

Modellelemente ausrichten (siehe Seite 120)

Optionen für das Erscheinungsbild von Diagrammen (siehe Seite 1284)

1.109 Modellelemente verschieben

Sie können Ihr eigenes Layout erstellen, indem Sie ein oder mehrere Diagrammelemente auswählen und an die gewünschte Position verschieben.

Sie können folgende Operationen durchführen:

- Wählen Sie ein Element aus, und ziehen Sie es an die neue Position.
- Wählen Sie mehrere Elemente aus, und ziehen Sie die gesamte Gruppe an die neue Position.
- Weisen Sie die Beziehungen manuell neu zu.

Anmerkung: Wenn Sie ein Element aus der Diagrammansicht herausziehen, wird im Diagramm automatisch ein Bildlauf durchgeführt, um dem Ziehen zu folgen.

Tip: Manuelle Layouts werden beim Schließen des Diagramms oder Projekts gespeichert und beim nächsten Öffnen automatisch wiederhergestellt. Sie bleiben nicht erhalten, wenn Sie einen der automatischen Layout-Befehle wählen (**Vollständiges Layout** oder **Größen optimieren**).

So verschieben Sie ein Element:

1. Wählen Sie das bzw. die zu verschiebenden Elemente aus.
2. Ziehen Sie das bzw. die ausgewählten Elemente an die Zielposition.

Tip: Klicken Sie mit der rechten Maustaste, und verwenden Sie die Befehle **Ausschneiden** und **Einfügen**. Sie können auch die Tastenkürzel für die Operationen Ausschneiden (STRG+X), Kopieren (STRG+C) und Einfügen (STRG+V) verwenden.

Siehe auch

Modellelemente auswählen (siehe Seite 132)

Tastenkürzel (siehe Seite 1278)

1.110 Diagramme umbenennen

Warnung: Das automatisch erzeugte Projekt-Namespace-Diagramm (Paket-Diagramm) kann nicht umgenannt werden.

So benennen Sie ein Diagramm um:

1. Doppelklicken Sie im Objektinspektor auf den Diagrammnamen, um den internen Editor zu starten.
2. Geben Sie einen neuen Namen ein.
3. Drücken Sie die EINGABETASTE.

Alternative:

1. Wählen Sie das Diagramm in der Modellansicht aus.
2. Drücken Sie die Taste F2, oder klicken Sie mit der rechten Maustaste, und wählen Sie im Kontextmenü den Befehl Umbenennen.
3. Geben Sie einen neuen Namen ein.
4. Drücken Sie die EINGABETASTE.

Ergebnis: Das Diagramm ist umbenannt.

Siehe auch

Diagramme erstellen (siehe Seite 116)

1.111 Beziehungen umlenken

So lenken Sie eine Beziehung um:

1. Wählen Sie eine Beziehung aus.
2. Ziehen Sie das Client- oder Anbieter-Ende der Beziehung zum gewünschten Zielobjekt.
3. Wenn Sie die Richtung der Beziehung ändern möchten, klicken Sie auf die Stelle der Beziehungslinie, an der die Umlenkung erfolgen soll.
4. Ziehen Sie die Linie. Together formt die Beziehung dann automatisch um.

Tip: Die Diagramm-Kontextmenüs von Modellelementen enthalten den Befehl **Layout>Alle Beziehungen routen**, mit dem die Beziehung optimiert werden kann.

Siehe auch

Überblick zu Modellelementen (siehe Seite 1448)

Diagramm-Layout festlegen (siehe Seite 127)

1.112 Größe von Modellelementen ändern

Die Größe von Diagrammelementen kann automatisch oder manuell geändert werden. Wenn einem Element, dessen Größe noch nie manuell geändert wurde, neue Komponenten hinzugefügt werden, wird dieses Element automatisch so vergrößert, dass die neuen Komponenten Platz finden.

So ändern Sie die Größe eines Elements manuell:

1. Klicken Sie auf ein Element. Das ausgewählte Element wird mit Punkten markiert.
2. Ziehen Sie einen der Punkte in die gewünschte Richtung.

Wenn sich der Inhalt eines Elements ändert, beispielsweise durch Hinzufügen oder Löschen von Komponenten, und deshalb nicht alle Komponenten im Element angezeigt werden können, werden an den rechten Bereichsseiten Bildlaufleisten eingeblendet.

So optimieren Sie die Größe eines Knotenelements:

1. Klicken Sie mit der rechten Maustaste auf ein Element.
2. Wählen Sie [Layout>Größe optimieren](#).

So optimieren Sie die Größe der Elemente im gesamten Diagramm:

1. Klicken Sie mit der rechten Maustaste auf den Diagrammhintergrund.
2. Wählen Sie [Layout>Größe optimieren](#).

Siehe auch

Automatisches Layout für Diagramme (siehe Seite 127)

1.113 Modellelemente auswählen

Zu den häufigsten Bearbeitungsoperationen für Diagrammelemente und Beziehungen zählen das Ziehen mit der Maus und das Ausführen von Befehlen in den Kontextmenüs von ausgewählten Elementen.

So wählen Sie ein Modellelement aus:

1. Öffnen Sie die Diagrammansicht.
2. Führen Sie in einem Diagramm folgende Aktionen durch:
 - Klicken Sie auf ein Element im Diagramm, um es auszuwählen.
 - Um mehrere Elemente auszuwählen, halten Sie die **STRG**-Taste gedrückt und klicken nacheinander auf die gewünschten Elemente.
 - Klicken Sie auf den Hintergrund, und ziehen Sie ein Lasso um einen Bereich, um alle darin enthaltenen Elemente auszuwählen.
 - Bei Elementen, die Komponenten enthalten, klicken Sie auf eine Komponente, um das Element auszuwählen.
 - Um alle Elemente in einem Diagramm auszuwählen, drücken Sie die Tastenkombination **STRG+A**. Alternativ klicken Sie mit der rechten Maustaste auf den Diagrammhintergrund und wählen im Kontextmenü den Befehl Alle auswählen.

Siehe auch

Modellelemente ausrichten ( siehe Seite 120)

Tastaturkürzel ( siehe Seite 1278)

1.114 Element-Stereotyp zuweisen

Die Zuweisung eines Stereotyps in einem Diagramm kann im internen Editor oder im Objektinspektor erfolgen.

Verwenden Sie eines der folgenden Verfahren, um ein Stereotyp zuzuweisen.

1. Zuweisen eines Stereotyps im internen Editor.
2. Zuweisen eines Stereotyps im Objektinspektor.

So weisen Sie ein Stereotyp im internen Editor zu:

1. Doppelklicken Sie auf den Namen des Stereotyps, um den internen Editor zu aktivieren.
2. Geben Sie den neuen Namen ein.
3. Drücken Sie die EINGABETASTE.

So weisen Sie ein Stereotyp im Objektinspektor zu:

1. Wählen Sie eine Klasse im Diagramm aus.
2. Wählen Sie im Objektinspektor das Feld Stereotyp aus.
3. Klicken Sie auf die Werteditor-Schaltfläche, und wählen Sie das gewünschte Stereotyp aus dem Kombinationsfeld. Sie können stattdessen auch den Stereotypnamen eingeben.

Ergebnis: Der Stereotypname wird in spitzen Klammern im Klassenknoten angezeigt.

Siehe auch

Das Profil "UML in Farbe" verwenden (siehe Seite 119)

1.115 Drag&Drop

Das Drag&Drop-Verfahren kann sowohl auf Member als auch auf Knotenelemente angewendet werden. Sie können Member (Methoden, Felder, Eigenschaften usw.) in der Diagrammansicht und in der Modellansicht per Drag&Drop verschieben und kopieren.

Drag&Drop-Aktionen zwischen der Modellansicht und der Diagrammansicht sowie innerhalb der Modellansicht funktionieren folgendermaßen:

- Wenn Sie ein Element in der Modellansicht auswählen, in das Diagramm ziehen und dort ablegen, wird eine Verknüpfung erzeugt.
- Wenn das Ziehen und Ablegen mit gedrückter **UMSCHALTTASTE** erfolgt, wird das Element in den ausgewählten Container verschoben.
- Erfolgt das Ziehen und Ablegen mit gedrückter **STRG-Taste**, wird eine Kopie des Elements im ausgewählten Container erzeugt.

Tip: Sie können das Drag&Drop-Verfahren auch verwenden, um den Ursprung und das Ziel von Beziehungen in Diagrammen zu ändern.

So verschieben Sie eine Beziehung an eine neue Position:

1. Wählen Sie eine Beziehung in der Diagrammansicht aus.
2. Zeigen Sie mit der Maus auf den Zielpfeil.
3. Ziehen Sie den Pfeil zur gewünschten Position. Wenn das Zielelement nicht auf dem Bildschirm angezeigt wird, ziehen Sie die Beziehung in die entsprechende Richtung. Im Diagramm wird dann ein Bildlauf durchgeführt.

Tip: Diese Anweisungen gelten auch für das Verschieben der Quelle einer Beziehung an eine andere (zulässige) Position.

Siehe auch

Modellelemente auswählen ( siehe Seite 132)

Modellelemente verschieben ( siehe Seite 128)

Tastaturkürzel ( siehe Seite 1278)

1.116 Benutzereigenschaften verwenden

Benutzereigenschaften werden mit dem Befehl Benutzereigenschaften erstellt. Dieser Befehl ist in den Kontextmenüs von Diagrammen und Diagrammelementen in der Diagrammansicht und in der Modellansicht enthalten. Nach der Erstellung werden die Benutzereigenschaften im Objektinspektor unter der Kategorie **Benutzereigenschaften** angezeigt und können bearbeitet werden.

So erstellen Sie Benutzereigenschaften:

1. Wählen Sie in der Diagrammansicht oder der Modellansicht das gewünschte Diagramm oder Modellelement aus.
2. Klicken Sie im Kontextmenü auf Benutzereigenschaften.
3. Klicken Sie im Dialogfeld Benutzereigenschaften hinzufügen/entfernen auf die Schaltfläche **Hinzufügen**. Der Eigenschaftsliste wird ein neuer Eintrag hinzugefügt, der aus den Feldern **Name** und **Wert** besteht.
4. Geben Sie den Namen und den Wert der Eigenschaft ein.
5. Stellen Sie die Liste der Benutzereigenschaften mit Hilfe der Schaltflächen **Hinzufügen** und **Entfernen** zusammen.
6. Klicken Sie auf OK, um den Vorgang abzuschließen.

Ergebnis: Die Kategorie **Benutzereigenschaften** wird im Objektinspektor angezeigt.

1.117 Diagramme als Grafik exportieren

So exportieren Sie ein Diagramm als Grafik:

1. Übergeben Sie in der Diagrammansicht den Fokus an das Diagramm, das exportiert werden soll.
2. Wählen Sie im Hauptmenü **Datei > Diagramm als Bild exportieren**.
3. Klicken Sie auf den Pfeil der Dropdown-Liste, um die Zoomeinstellungen der Diagrammgrafik anzuzeigen und anzupassen.
4. Klicken Sie auf **Speichern**. Der Datei-Browser wird geöffnet.
5. Wechseln Sie zu dem Verzeichnis, in dem Sie die Grafikdatei speichern möchten.
6. Geben Sie einen Namen ein. Per Voreinstellung erhält die Grafikdatei den Namen des Diagramms in RAD Studio.
7. Wählen Sie ein Grafikformat aus.
8. Klicken Sie auf **Speichern**.

Siehe auch

Überblick zum Importieren und Exportieren (siehe Seite 1459)

1.118 Beziehungslinien mit Umlenkpunkten erstellen

Wenn das Diagramm viele Elemente enthält, die dicht beieinander liegen, können Sie zwischen Quell- und Zielelement eine Beziehungslinie mit Umlenkpunkten zeichnen und auf diese Weise andere Elemente umgehen.

So erstellen Sie eine Beziehungslinie mit Umlenkpunkten:

1. Klicken Sie in der Tool-Palette auf die Schaltfläche **Beziehung**.
2. Klicken Sie auf das Quellelement.
3. Ziehen Sie die Beziehungslinie, und klicken Sie an Stellen, an denen ein Linienabschnitt definiert werden soll, mit der Maus. Die Abschnitte einer Beziehungslinie sind durch zwei blaue Punkte gekennzeichnet. Die Punkte werden immer angezeigt, wenn Sie die Beziehung im Diagramm auswählen.
4. Klicken Sie auf das Zielelement, um die Beziehung abzuschließen.

Tip: Sobald eine Beziehung erstellt ist, können Sie ihr Umlenkpunkte hinzufügen. Wählen Sie die Beziehung im Diagramm aus, und ziehen Sie bis zur gewünschten Position. In der folgenden Abbildung ist diese Technik dargestellt.

Siehe auch

Beziehungen umlenken (siehe Seite 130)

Eine einfache Beziehung erstellen (siehe Seite 141)

Klassendiagrammbeziehungen (siehe Seite 1296)

1.119 Mehrere Elemente erstellen

Sie können mehrere Elemente desselben Typs in ein Diagramm einfügen, ohne zur Tool-Palette zurückzukehren oder das Kontextmenü des Diagramms zu verwenden. Jedes Element hat einen vorgegebenen Namen, der mit dem internen Editor oder im Objektinspektor bearbeitet werden kann.

So erstellen Sie mehrere Elemente:

1. Halten Sie die Taste **UMSCHALT** gedrückt, klicken Sie in der Tool-Palette auf die Schaltfläche für das Element, das erstellt werden soll (die Schaltfläche bleibt aktiviert). Lassen Sie die Taste **UMSCHALT** los.
2. Klicken Sie auf die gewünschte Position im Diagrammhintergrund. Das neue Element wird an der Stelle, an der Sie mit der Maus klicken, in das Diagramm eingefügt.
3. Klicken Sie auf die nächste Position im Diagrammhintergrund. Das nächste Element wird in das Diagramm eingefügt.
4. Wiederholen Sie die Aktion für alle Elemente dieses Typs, die eingefügt werden sollen.
5. Wenn Sie die Elementerstellung beenden möchten, klicken Sie auf die **Zeiger**-Schaltfläche in der Tool-Palette oder drücken die Taste **ESC**. Dadurch wird die Auswahl des Elements aufgehoben, nachdem der interne Editor des zuletzt eingefügten Elements geschlossen wurde.

Tip: Nach einer Auswahl in der Tool-Palette oder der ersten Aktion zum Zeichnen oder Platzieren mehrerer Elemente können Sie die Operation abbrechen, indem Sie auf die **Zeiger**-Schaltfläche in der Tool-Palette klicken oder die Taste **ESC** drücken.

Siehe auch

Ein Element erstellen ( siehe Seite 142)

Eine einfache Beziehung erstellen ( siehe Seite 141)

Tastaturkürzel ( siehe Seite 1278)

1.120 Verknüpfungen erstellen

Sie können eine Verknüpfung zu einem Modellelement auf dem Diagrammhintergrund erstellen. Dies kann auf drei Arten geschehen:

- Im Dialogfeld Verknüpfungen bearbeiten in der Diagrammansicht.
- Durch Kopieren und Einfügen in der Modellansicht.
- Mit dem Befehl Als Verknüpfung hinzufügen im Kontextmenü der Modellansicht.

Verwenden Sie eines der folgenden Verfahren, um eine Verknüpfung zu erstellen:

1. Erstellen einer Verknüpfung im Dialogfeld Verknüpfungen bearbeiten.
2. Erstellen einer Verknüpfung per Drag&Drop.
3. Erstellen einer Verknüpfung durch Kopieren und Einfügen.
4. Erstellen einer Verknüpfung über das Kontextmenü der Modellansicht.

So erstellen Sie eine Verknüpfung mit Hilfe des Dialogfelds `color="ide"Verknüpfungen bearbeiten`:

1. Klicken Sie mit der rechten Maustaste auf den Diagrammhintergrund.
2. Klicken Sie im Kontextmenü auf **Hinzufügen**▶**Verknüpfungen**.

Tip: Sie können stattdessen auch die Tastenkombination STRG+UMSCHALT+M drücken, um das Dialogfeld Verknüpfungen bearbeiten zu öffnen.

3. Wählen Sie im Dialogfeld Verknüpfungen bearbeiten das gewünschte Element in der hierarchischen Anzeige mit den verfügbaren Elementen aus.
4. Klicken Sie auf Hinzufügen, um das ausgewählte Element in die Liste **Existierende und/oder zum Hinzufügen bereite Elemente** aufzunehmen.
5. Wenn Sie alle gewünschten Elemente zur Liste hinzugefügt haben, klicken Sie auf OK.

So erstellen Sie eine Verknüpfung per Drag&Drop:

1. Wählen Sie das Element in der Modellansicht aus.
2. Ziehen Sie das Element in das Diagramm, und lassen Sie die Maustaste los.

So erstellen Sie eine Verknüpfung durch Kopieren und Einfügen:

1. Klicken Sie in der Modellansicht mit der rechten Maustaste auf das Element, das als Referenz in das aktuelle Diagramm aufgenommen werden soll.
2. Wählen Sie im Kontextmenü den Befehl Kopieren.
3. Klicken Sie mit der rechten Maustaste in das Zieldiagramm, und wählen Sie im Kontextmenü Verknüpfung einfügen.

Tip: Sie können auch ein Element in einem Diagramm kopieren und es als Verknüpfung in ein anderes Diagramm einfügen.

So erstellen Sie eine Verknüpfung über das Kontextmenü der `color="ide"Modellansicht`.

1. Öffnen Sie das Diagramm, in das die Verknüpfung eingefügt werden soll.
2. Wählen Sie in der Modellansicht das Element aus, das als Verknüpfung in das aktuelle Diagramm aufgenommen werden soll.
3. Klicken Sie in der Modellansicht mit der rechten Maustaste auf das Element, und wählen Sie im Kontextmenü Als

Verknüpfung hinzufügen.

Siehe auch

Überblick zu Verknüpfungen (siehe Seite 1449)

Überblick zu Hyperlinks (siehe Seite 1450)

Ein Element erstellen (siehe Seite 142)

1.121 Eine einfache Beziehung erstellen

In einem Designprojekt können Sie eine Beziehung zu einem anderen Knoten bzw. zu einer Verknüpfung eines Elements desselben oder eines anderen Designprojekts erstellen (die Projekte müssen dieselbe UML-Version haben).

In einem Implementierungsprojekt können Sie eine Beziehung zu einem anderen Knoten oder zu einer Verknüpfung eines Elements desselben Projekts erstellen.

So erstellen Sie eine einfache Beziehung zwischen zwei Knoten:

1. Klicken Sie in der Tool-Palette des Diagramms auf die Schaltfläche für den Beziehungstyp, der im Diagramm erzeugt werden soll. Die Schaltfläche wird gedrückt dargestellt.
2. Klicken Sie auf das Quellelement.
3. Ziehen Sie mit der Maus zum Zielement, und lassen Sie die Maustaste los, sobald das zweite Element optisch hervorgehoben wird.

Siehe auch

Beziehungen umlenken ( siehe Seite 130)

Beziehungslinien mit Umlenkpunkten erstellen ( siehe Seite 137)

Beziehungen nach Pattern erstellen ( siehe Seite 226)

Klassendiagrammbeziehungen ( siehe Seite 1296)

1.122 Einzelnes Modellelement erstellen

So erstellen Sie ein einzelnes Modellelement:

1. Öffnen Sie ein Zieldiagramm in der Diagrammansicht.
2. Wählen Sie im Menü **Ansicht** den Befehl Tool-Palette.
3. Aktivieren Sie in der Tool-Palette die Registerkarte UML [Diagrammtyp], um die verfügbaren Modellelemente anzuzeigen.
4. Klicken Sie in der Tool-Palette auf das Symbol des Elements, das in das Diagramm eingefügt werden soll. Die Schaltfläche wird gedrückt dargestellt.

Tip: Die Symbole sind mit Beschriftungen versehen.

5. Klicken Sie an der Stelle auf den Diagrammhintergrund, an der das neue Element platziert werden soll. Das neue Element wird erstellt, und der interne Editor für die Vergabe eines Namens wird geöffnet.

Tip: Alternativ klicken Sie mit der rechten Maustaste auf den Diagrammhintergrund und wählen im Kontextmenü den Befehl Hinzufügen. Im Untermenü werden alle Basiselemente, die in das Diagramm eingefügt werden können, sowie der Befehl Verknüpfungen angezeigt.

Siehe auch

Mehrere Modellelemente erstellen (siehe Seite 138)

Eine einfache Beziehung erstellen (siehe Seite 141)

1.123 Diagramme drucken

Sie können Diagramme einzeln oder gruppenweise drucken. Es ist auch möglich, alle Diagramme in einem Projekt auszudrucken.

So drucken Sie ein Diagramm:

1. Zeigen Sie das Diagramm in der Diagrammansicht an, und wählen Sie im Hauptmenü **Datei** **Drucken**. Das Dialogfeld **Diagramm drucken** wird geöffnet.
2. Legen Sie im Listenfeld **Diagramme drucken** den Diagrammbereich fest, der gedruckt werden soll:
 - **Aktives Diagramm:** Das aktuell ausgewählte Diagramm wird gedruckt.
 - **Aktives mit benachbarten:** Das aktuelle Diagramm und die anderen Diagramme desselben Projekts werden gedruckt.
 - **Alle geöffneten:** Alle aktuell in der Diagrammansicht geöffneten Diagramme werden gedruckt.
 - **Alle im Modell:** Alle Diagramme innerhalb einer Projektgruppe werden gedruckt.
3. Im Feld **Druck-Zoom** geben Sie den Zoom-Faktor an.
4. Passen Sie ggf. die Seiten- und Druckereinstellungen an:
 - Klicken Sie auf das Listenfeld **Drucken**, und wählen Sie **Druckdialog**, um den Zieldrucker auszuwählen.
 - Im Dialogfeld **Optionen** (**Together** **►** **(Ebene)** **►** **Diagramm** **►** **Drucken**) können Sie das Papierformat, die Ausrichtung und die Ränder festlegen.

Tip: Klicken Sie auf **Druckbild**, um das Vorschaufenster zu öffnen. Verwenden Sie den Schieberegler **Vorschau-Zoom** und das Kontrollkästchen **Autom. Zoom der Vorschau** nach Bedarf.

Siehe auch

Druckoptionen für Diagramme (siehe Seite 1287)

1.124 Diagrammdateien der Versionskontrolle unterstellen

Sie können in Together Ihr Modell auf die gleiche Weise wie andere Projektressourcen mit der Versionskontrolle überwachen.

Die Diagrammelemente sind Bestandteil der übergeordneten Standard-Paket- bzw. Namespace-Dateien (default.txaPackage). Wenn Sie durch die Bearbeitung eines Diagramms Elemente hinzufügen oder löschen und die Elementeigenschaften ändern, müssen Sie deshalb sowohl die Diagrammdatei als auch die entsprechende default.txaPackage-Datei ein- und auschecken. Wird lediglich die Größe und Platzierung von Elementen geändert, reicht es aus, die Diagrammdatei ein- und auszuchecken. Um unsynchronisierte Ergebnisse und Konflikte zu vermeiden, sollten alle Team-Mitglieder, die am selben Projekt arbeiten, das gesamte Projekt auschecken und mit einer Exklusiv-Sperre versehen.

Mit Hilfe des Befehls Historie können Diagrammdateien auf Änderungen überprüft werden.

Warnung: Dieses Thema befasst sich mit modellierungsspezifischen Operationen der Versionskontrolle. Ausführliche Informationen zur Versionskontrolle finden Sie in der RAD Studio-Dokumentation bzw. in der Dokumentation des verwendeten Versionskontrollsystems.

So unterstellen Sie Diagrammdateien der Versionskontrolle:

1. Stellen Sie sicher, dass auf dem Computer eine Integration für ein unterstütztes Versionskontrollsystem installiert ist. Dies ist die Voraussetzung für den Zugriff auf die Versionskontrollbefehle.
2. Falls noch nicht geschehen, fügen Sie Ihr Projekt (bzw. die Projektgruppe) der Versionskontrolle hinzu.
3. Rufen Sie Versionskontrollbefehle für die Diagrammdateien auf. Dies kann auf zwei Arten erfolgen:
 - StarTeam. Die Befehle dieses Menüs ermöglichen es Ihnen, Together-Diagrammdateien in das Repository des Versionskontrollsystems zu übertragen. Sie können Dateien ein- und auschecken, holen, ausschließen, vergleichen und das Auschecken von Dateien rückgängig machen.
 - Über das Kontextmenü der Projektverwaltung. Mit den Befehlen dieses Kontextmenüs können Sie die oben genannten Operationen ausführen und außerdem das gesamte Projekt bzw. die gesamte Projektgruppe ein- und auschecken und mit den Standard-Paket/Namespace-Dateien arbeiten.

Anmerkung: Nachdem ein Projekt (eine Projektgruppe) der Versionskontrolle unterstellt wurde, enthalten die Diagrammansicht und die Projektverwaltung Symbole, die den Status der Modellelemente unter Versionskontrolle anzeigen.

So schließen Sie Dateien aus der Versionskontrolle aus:

1. Öffnen Sie das Diagramm in der Diagrammansicht.
2. Wählen Sie StarTeam. Der Befehl wirkt sich auf das Diagramm aus, das in der Diagrammansicht geöffnet ist und den Fokus besitzt.

So checken Sie ein Projekt bzw. eine Projektgruppe ein und aus:

1. Öffnen Sie die Projektverwaltung.
2. Klicken Sie mit der rechten Maustaste auf den Projektknoten oder den Projektgruppenknoten.
3. Wählen Sie im Kontextmenü Einchecken oder Auschecken.

So checken Sie Diagramme ein und aus:

1. Öffnen Sie das Diagramm in der Diagrammansicht, und wählen Sie im Hauptmenü StarTeam ▶ Auschecken. Das Dialogfeld zum Auschecken von Dateien wird geöffnet.

Tip: Um eine schreibgeschützte Kopie der letzten Version des Diagramms abzurufen, wählen Sie im Hauptmenü **StarTeam**►???.

2. Klicken Sie auf Auschecken.
3. Wenn die Arbeit mit der Diagrammdatei abgeschlossen ist, wählen Sie im Hauptmenü **StarTeam**►**Einchecken**.
4. Das Dialogfeld zum Einchecken von Dateien wird geöffnet.
5. Klicken Sie auf Einchecken.

Mit dem Befehl Einchecken wird das Diagramm gesperrt. Diagrammelemente in gesperrten Diagrammen werden mit kleinen Sperrsymbolen dargestellt. Sie können das gesperrte Diagramm bearbeiten. Sobald Sie Elemente ändern, hinzufügen oder löschen, wird automatisch das Dialogfeld zum Auschecken angezeigt. Danach ist das Diagramm nicht mehr gesperrt.

So machen Sie das Auschecken rückgängig:

1. Wenn Sie nach dem Auschecken eines Diagramms keine Änderung daran vornehmen, können Sie den Befehl **Rückgängig: Auschecken** aufrufen. Dadurch wird das Auschecken rückgängig gemacht, und die editierbare Version der Datei wird aus dem Arbeitsordner entfernt. Ihre lokale Kopie wird durch die neueste Version der Diagrammdatei im Versionskontroll-Repository überschrieben. Änderungen, die Sie seit dem letzten Auschecken der Datei an der lokalen Kopie vorgenommen haben, gehen verloren.
2. Öffnen Sie das Diagramm in der Diagrammansicht.
3. Wählen Sie **StarTeam**►**Rückgängig**. Der Befehl wirkt sich auf das Diagramm aus, das in der Diagrammansicht geöffnet ist und den Fokus besitzt.

So vergleichen Sie Diagrammversionen:

1. Öffnen Sie das Diagramm in der Diagrammansicht.

Warnung: Um die Versionen einer Standard-Paketdatei auf Unterschiede zu prüfen, wählen Sie zunächst den entsprechenden `default.txaPackage`-Knoten in der Projektverwaltung aus.

2. Wählen Sie im Hauptmenü **StarTeam**►**Inhalt vergleichen**. Ein Dialogfeld wird geöffnet.
3. Wählen Sie eine Version in der Liste aus, und klicken Sie auf **Diff**. Das Fenster **Visual Diff** wird geöffnet.

Tip:

Siehe auch

Diagramme erstellen (siehe Seite 116)

1.125 In Diagrammen suchen

In Together können Sie mit den Suchen- und Ersetzen-Funktionen von RAD Studio Modellelemente in Modelldiagrammen suchen.

So suchen Sie in Diagrammen:

1. Wählen Sie **Suchen ▶ (Suchbefehl)**, um die Suchen- und Ersetzen-Funktionen von RAD Studio aufzurufen.
2. Sie können in einem bestimmten Bereich nach beliebigen Zeichenfolgen suchen. Außerdem können Sie nach ganzen Wörtern oder Teilsätzen suchen, Platzhalter und reguläre Ausdrücke in den Suchbegriffen verwenden und festlegen, dass die Groß-/Kleinschreibung berücksichtigt wird.
3. Überprüfen Sie die Suchergebnisse.

Siehe auch

Quelltext nach Verwendungen durchsuchen (siehe Seite 147)

1.126 Quelltext nach Verwendungen durchsuchen

Sie können in Together nicht nur in Diagrammen suchen, sondern auch überprüfen, wie ein Element oder Member in einem Quelltextprojekt verwendet wird. Im Dialogfeld Verwendung suchen können Sie in Implementierungsprojekten nach Referenzen und Überschreibungen von Elementen und Membern suchen.

Das Dialogfeld kann in der Diagrammansicht oder der Modellansicht mit dem Befehl **Verwendung suchen** im Kontextmenü eines Elements geöffnet werden. Für Designprojekte steht dieser Befehl nicht zur Verfügung.

So durchsuchen Sie Quelltext nach Verwendungen eines Elements:

1. Klicken Sie mit der rechten Maustaste auf ein Element oder einen Namespace, und wählen Sie im Kontextmenü Verwendung suchen. Das Dialogfeld **Verwendung suchen** wird geöffnet und das ausgewählte Element im Bereich **Zu suchendes Element** angezeigt.
2. Aktivieren Sie im Bereich **Optionen** die gewünschten Optionen:
 - Verwendung des Elements
 - Verwendung der Member
 - Verwendung von deklarierten Klassen
 - Implementierungen
 - Überschreiben
 - Using/Import einbeziehen
 - Self überspringen
3. Klicken Sie auf **Suchen**.

Das Suchergebnis wird in hierarchischer Form auf einer Registerkarte im Fenster **Verwendung suchen** angezeigt. Jeder Knoten enthält sämtliche Verwendungen des Elements in einer bestimmten Klasse. Bei jeder neuen Suche wird eine weitere Registerkarte in das Fenster eingefügt.

Die Symbolleiste des Fensters **Verwendung suchen** enthält Schaltflächen, mit denen Sie den Inhalt der Knoten ein- oder ausblenden und die Suche auf der aktiven Registerkarte mit denselben Einstellungen erneut starten können.

Das Kontextmenü für eine Registerkarte mit den Suchergebnis enthält folgende Befehle:

Befehl
Schließen
Alle schließen
Alle außer diesem schließen

Siehe auch

In Diagrammen suchen (siehe Seite 146)

1.127 Aktivität für einen Zustand erzeugen

So erzeugen Sie eine Aktivität für einen Zustand:

1. Öffnen Sie die Diagrammansicht.
2. Klicken Sie mit der rechten Maustaste auf einen Zustand, und wählen Sie im Kontextmenü **Hinzufügen > Aktivität**.
Ergebnis: In dem Zustand wird eine neue Aktivität erzeugt.

Siehe auch

UML 1.5-Aktivitätsdiagramm (siehe Seite 1308)

1.128 UML 1.5-Aktivitätsdiagramme erstellen

Folgende Tipps und Techniken helfen Ihnen bei der Erstellung eines UML 1.5-Aktivitätsdiagramm.

Gehen Sie zur Erstellung eines UML 1.5-Aktivitätsdiagramms folgendermaßen vor:

1. Erstellen Sie einen oder mehrere Verantwortlichkeitsbereiche. Sie können mehrere Verantwortlichkeitsbereiche in ein Diagramm einfügen oder für jeden Bereich ein eigenes Diagramm erstellen.

Warnung: Verantwortlichkeitsbereiche können nicht verschachtelt werden.

2. Erstellen Sie eine oder mehrere Aktivitäten. Sie können mehrere Aktivitäten in einen Verantwortlichkeitsbereich einfügen oder für jede Aktivität einen eigenen Bereich erstellen.

Warnung: Aktivitäten können nicht verschachtelt werden.

3. Sie können bequemer navigieren, wenn Sie zuerst den Hauptablauf der Aktivitäten modellieren. Erstellen Sie danach Verzweigungen, parallele Abläufe und den Objektfluss.

Tip: Verwenden Sie gegebenenfalls getrennte Diagramme, und verknüpfen Sie sie durch Hyperlinks.

4. Erstellen Sie Start-, Ende-, Signalempfangs- und Signalsendungselemente für die Verantwortlichkeitsbereiche. Wenn die Aktivität mehrere Startpunkte hat, können diese gleichzeitig verwendet werden.

5. Erstellen Sie Objektknoten. Zwischen Objektknoten und Klassen in Klassendiagrammen werden keine Beziehungen erstellt. Sie können aber Hyperlinks verwenden, um die Interpretation der Diagramme zu erleichtern.

6. Erzeugen Sie Zustandsknoten für die Verantwortlichkeitsbereiche.

Tip: Zustände können verschachtelt werden.

7. Erstellen Sie bei Bedarf einen Historienknoten.

8. Verbinden Sie die Knoten über Beziehungen.

9. Wahlweise können Sie auch Verknüpfungen zu Elementen in anderen Diagrammen erstellen.

Siehe auch

Verknüpfungen erstellen ( siehe Seite 139)

UML 1.5-Aktivitätsdiagramme – Referenz ( siehe Seite 1308)

1.129 Klassifizierer instantiiieren

Sie können in einem UML 1.5-Designprojekt ein Objekt erstellen, das eine Klasse oder ein Interface aus demselben oder einem anderen UML 1.5-Designprojekt oder aus einem Implementierungsprojekt in derselben Projektgruppe instantiiert. In einem Implementierungsprojekt können Sie ein Objekt erstellen, das eine Klasse oder ein Interface aus demselben Projekt, aus einem UML 1.5-Designprojekt oder aus einem referenzierten Projekt instantiiert. Derartige Beziehungen lassen sich im Objektinspektor oder über Abhängigkeitsbeziehungen zu Verknüpfungen erzeugen.

So instantiiieren Sie einen Klassifizierer:

1. Wählen Sie in einem UML 1.5-Klassendiagramm ein Objekt aus.
2. Wählen Sie im Objektinspektor das Feld Instantiates aus.
3. Klicken Sie auf die Auswahlabschaltfläche. Das Dialogfeld Auswahl des zu instanzierenden Typs wird angezeigt.
4. In diesem Dialogfeld können Sie einen Klassifizierer (Klasse oder Interface) auswählen.

Tip: Alternativ erstellen Sie eine **Abhängigkeitsbeziehung** von diesem Objekt zu einem Klassifizierer oder seiner Verknüpfung.

Siehe auch

UML 1.5-Klassendiagramm (siehe Seite 1294)

1.130 UML 1.5-Komponentendiagramme erstellen

Folgende Tipps und Techniken helfen Ihnen bei der Arbeit mit UML 1.5-Komponentendiagrammen. Es kann hilfreich sein, die Erstellung eines Modells mit Komponentendiagrammen zu beginnen. Dies gilt besonders bei der Modellierung eines großen Systems, beispielsweise eines verteilten Client-Server-Softwaresystems mit zahlreichen verknüpften Modulen. Verwenden Sie Komponentendiagramme zur Modellierung der logischen Struktur und Verteilungsdiagramme zur Modellierung der physischen Struktur des Systems.

Gehen Sie zur Erstellung eines UML 1.5-Komponentendiagramms folgendermaßen vor:

1. Erstellen Sie eine Hierarchie von Subsystemen.

Tip: Subsysteme können verschachtelt werden.

2. Erstellen Sie eine Hierarchie von Komponenten. Die größte Komponente kann das gesamte System oder dessen Hauptbestandteil sein (beispielsweise *Server-Anwendung*, *IDE* oder *Service*).

Tip: Komponentenknoten können verschachtelt werden. Es gibt zwei Verfahren für die Erstellung eines verschachtelten Komponentenknotens: Sie können eine vorhandene Komponente auswählen und eine untergeordnete Komponente einfügen. Alternativ können Sie zwei separate Komponenten erstellen und sie über eine Assoziations-Kompositions-Beziehung miteinander verbinden.

3. Erstellen Sie Interfaces. Jede Komponente kann über ein Interface verfügen.
4. Erzeugen Sie Beziehungen zwischen Elementen.
5. Wahlweise können Sie auch Verknüpfungen zu Elementen in anderen Diagrammen erstellen.

Siehe auch

[Verknüpfungen erstellen](#) (siehe Seite 139)

[UML 1.5-Komponentendiagramme – Referenz](#) (siehe Seite 1310)

1.131 UML 1.5-Verteilungsdiagramme erstellen

Folgende Tipps und Techniken helfen Ihnen bei der Erstellung eines UML 1.5-Verteilungsdiagramms. Es kann hilfreich sein, die Erstellung eines Modells mit Verteilungsdiagrammen zu beginnen. Dies gilt besonders bei der Modellierung eines großen Systems mit zahlreichen Modulen, die sich auf verschiedenen Computern befinden. Verwenden Sie Verteilungsdiagramme zur Modellierung der physischen Struktur und Komponentendiagramme zur Modellierung der logischen Struktur des Systems.

Gehen Sie zur Erstellung eines UML 1.5-Verteilungsdiagramms folgendermaßen vor:

1. Erstellen Sie eine Hierarchie von Knoten.

Tip: Knoten können verschachtelt werden.

2. Erstellen Sie eine Hierarchie von Komponenten. Die größte Komponente kann das gesamte System oder dessen Hauptbestandteil sein (beispielsweise *Server-Anwendung*, *IDE* oder *Service*).

Tip: Komponenten können verschachtelt werden. Es gibt zwei Verfahren für die Erstellung einer verschachtelten Komponente: Sie können eine vorhandene Komponente auswählen und eine untergeordnete Komponente einfügen. Alternativ können Sie zwei separate Komponenten erstellen und sie über eine Assoziations-Kompositions-Beziehung miteinander verbinden.

3. Stellen Sie dar, wie Komponenten in Knoten angeordnet sind. Hierfür stehen zwei Möglichkeiten zur Verfügung:

- Erstellen Sie eine Support-Beziehung zwischen der Komponente und dem Knoten. Eine Support-Beziehung ist eine Abhängigkeitsbeziehung, deren Stereotyp-Feld auf **support** gesetzt ist.
- Verschachteln Sie die Komponente grafisch im Knoten.

4. Erstellen Sie gegebenenfalls Objekte.

5. Erstellen Sie Interfaces. Jede Komponente kann über ein Interface verfügen.

6. Weisen Sie auf eine vorübergehende Beziehung zwischen einer Komponente und einem Knoten hin. Objekte und Komponenten können von einer Komponenteninstanz bzw. Knoteninstanz zu einer anderen migrieren. In diesem Fall befindet sich das Objekt (die Komponente) nur vorübergehend in der Komponente (dem Knoten). Zeigen Sie dies durch eine Abhängigkeitsbeziehung mit dem Stereotyp **becomes** an.

7. Wahlweise können Sie auch Verknüpfungen zu Elementen in anderen Diagrammen erstellen.

Siehe auch

[Verknüpfungen erstellen](#) (siehe Seite 139)

[UML 1.5-Verteilungsdiagramme – Referenz](#) (siehe Seite 1311)

1.132 Konditionale Blöcke hinzufügen

Anmerkung: Wenn die Kontrollstruktur eine Bedingung erfordert, geben Sie diese mit dem internen Editor ein, oder verwenden Sie im Objektinspektor das Feld **Condition**.

So fügen Sie der Aktivierungsleiste einen Anweisungsblock hinzu:

1. Klicken Sie in der Tool-Palette auf die Schaltfläche **Konditionaler Block**.
2. Klicken Sie auf die Aktivierungsleiste, die als Ziel fungiert.

Alternative:

1. Klicken Sie in einem Sequenzdiagramm mit der rechten Maustaste auf eine Aktivierungsleiste.
2. Wählen Sie im Kontextmenü **Hinzufügen > Konditionaler Block**.

So legen Sie den Typ des konditionalen Blocks fest (if, for usw.):

1. Öffnen Sie den Objektinspektor.
2. Klicken Sie auf den Pfeil der Dropdown-Liste, und treffen Sie eine Auswahl.

Siehe auch

UML 1.5-Interaktionsdiagramme (siehe Seite 1302)

1.133 Objekten einen Klassifizierer zuordnen

In Sequenz- oder Kollaborationsdiagrammen können Sie Zuordnungen zwischen Objekten (in einem Interaktionsdiagramm) und Klassifizierern (in einem Klassendiagramm) erstellen. Sie können Klassen für ein Objekt im Modell auswählen oder neue Klassen erstellen und dem Modell hinzufügen.

Ein Objekt kann Klassifizierer der verschiedenen Quelltextprojekte in einer Projektgruppe instantiiieren, sofern auf diese Projekte im aktuellen Projekt verwiesen wird.

Welche Klassifizierer verfügbar sind, hängt vom Projekttyp ab:

- **Designprojekte:** Klassen, Interfaces
 - **C#-Implementierungsprojekte:** Klassen, Interfaces, Strukturen
- *

So ordnen Sie einem Objekt einen vorhandenen Klassifizierer zu:

1. Wählen Sie das Objekt aus.
2. Wählen Sie im Kontextmenü des Objekts den Befehl Klasse auswählen.
3. Ein Untermenü mit einer Liste der verfügbaren Klassifizierer wird angezeigt. Wenn sich der gewünschte Klassifizierer nicht in der Liste befindet, klicken Sie auf **Weitere**, um die Modellhierarchie anzuzeigen.
4. Das Dialogfeld Typ zum Instantiiieren auswählen wird geöffnet. Wählen Sie im Modell den Klassifizierer aus, und klicken Sie auf OK.

Tip: Sie können auch den Objektinspektor verwenden. Klicken Sie auf das Feld **Instantiates**, und wählen Sie den Klassifizierer im Modell aus.

Ergebnis: Der vollständige Pfadname des instantiierten Klassifizierers wird im Objekt angezeigt.

Tip: Um einem Objekt einen Klassifizierer aus einem anderen Projekt zuzuordnen, fügen Sie dieses als referenziertes Projekt hinzu.

So erstellen Sie einen neuen Klassifizierer für ein vorhandenes Objekt:

1. Wählen Sie das Objekt aus.
2. Klicken Sie im Kontextmenü auf Hinzufügen.
3. Wählen Sie im Untermenü den gewünschten Klassifizierertyp aus.

Ergebnis: Der neue Klassifizierer wird dem Modell hinzugefügt. Im Interaktionsdiagramm wird eine Verknüpfung für den neuen Klassifizierer angezeigt, die mit dem Objekt durch eine Abhängigkeitsbeziehung verbunden ist.

So heben Sie die Zuordnung eines Objekts auf:

1. Wählen Sie das Objekt aus.
2. Wählen Sie im Kontextmenü des Objekts den Befehl Klassenzuordnung aufheben.

Ergebnis: Die Zuordnung wird entfernt, der Klassifizierer ist jedoch weiterhin im Modell vorhanden.

So navigieren Sie zwischen Klassifizierern und Objekten:

1. Wählen Sie das Objekt im Diagramm aus.
2. Klicken Sie mit der rechten Maustaste, und wählen Sie im Kontextmenü Modellansicht synchronisieren, um den Fokus auf

den Klassifizierer in der Modellansicht zu platzieren. Sie können aber auch Zur Klassendefinition wechseln wählen, um den Klassifizierer im Quelltext zu öffnen (bei Implementierungsprojekten).

So erstellen Sie eine Verknüpfung zu einem Klassifizierer in einem Interaktionsdiagramm:

1. Wählen Sie im Diagramm ein Objekt aus, das einen Klassifizierer instantiiert.
2. Klicken Sie mit der rechten Maustaste, und wählen Sie im Kontextmenü Klasse importieren.

Ergebnis: Eine Verknüpfung mit dem instantiierten Klassifizierer wird in das Diagramm eingefügt.

Siehe auch

Mit referenzierten Projekten arbeiten ( siehe Seite 266)

UML 1.5-Klassendiagramm ( siehe Seite 1294)

UML 1.5-Interaktionsdiagramme ( siehe Seite 1302)

1.134 Nachrichtenbeziehungen verzweigen

Anleitung zum Verzweigen von Nachrichten, die an derselben Position auf der **Lebenslinie** beginnen.

So verzweigen Sie eine Nachrichtenbeziehung:

1. Wählen Sie die Nachrichtenbeziehung im Sequenz- oder Kollaborationsdiagramm aus.
2. Klicken Sie mit der rechten Maustaste auf die Nachrichtenbeziehung, und wählen Sie im Kontextmenü **Verzweigung** ▾ **Mit vorherigem**.

So entfernen Sie eine Verzweigung:

1. Wählen Sie die Nachrichtenbeziehung aus, deren Verzweigung entfernt werden soll.
2. Klicken Sie mit der rechten Maustaste auf die Nachrichtenbeziehung, und wählen Sie im Kontextmenü **Verzweigung** ▾ **Ohne**.

Siehe auch

Mit UML 1.5-Nachrichten arbeiten (↗ siehe Seite 158)

UML 1.5-Interaktionsdiagramme (↗ siehe Seite 1302)

1.135 Zwischen UML 1.5-Sequenz- und -Kollaborationsdiagrammen umschalten

Sie können Sequenz- in Kollaborationsdiagramme umwandeln (und umgekehrt). Wenn Sie jedoch ein neues Diagramm erstellen, müssen Sie festlegen, ob es sich um ein Sequenz- oder ein Kollaborationsdiagramm handelt.

So führen Sie die Umwandlung zwischen Sequenz- und Kollaborationsdiagrammen durch:

1. Klicken Sie mit der rechten Maustaste auf den Diagrammhintergrund.
2. Wählen Sie bei einem Sequenzdiagramm im Kontextmenü den Befehl Als Kollaboration anzeigen. Bei einem Kollaborationsdiagramm wählen Sie Als Sequenz anzeigen.
3. Wiederholen Sie diesen Schritt, um hin- und herzuwechseln.

Nachdem Sie ein Sequenzdiagramm zum ersten Mal in ein Kollaborationsdiagramm umgewandelt oder zwischen den Umwandlungen neue Objekte in das Sequenzdiagramm eingefügt haben, sollten Sie ein Layout für das gesamte Diagramm durchführen.

Siehe auch

UML 1.5-Interaktionsdiagramme (siehe Seite 1302)

1.136 Mit UML 1.5-Nachrichten arbeiten

In diesem Abschnitt erfahren Sie, wie Sie in Sequenz- und Kollaborationsdiagrammen mit Nachrichten arbeiten können. Obwohl diese beiden Diagrammtypen äquivalent sind, unterscheiden sie sich doch hinsichtlich der Verwendung von Nachrichten.

In einem Kollaborationsdiagramm werden alle Nachrichtenbeziehungen zwischen zwei Objekten als Beziehungslinie mit einer darüber befindlichen Liste der Nachrichten angezeigt. Die Beziehungslinie wird so lange angezeigt, wie mindestens eine Nachricht zwischen den Objekten vorhanden ist. Die Nachrichten sind in der Liste entsprechend ihrem zeitlichen Auftreten aufgeführt. Die erste Nachricht befindet sich oben. Sie können außer diesen Nachrichtenbeziehungen auch Assoziations- und Aggregationsbeziehungen hinzufügen. Diese Beziehungen sind aber nicht sichtbar, wenn Sie das Diagramm als Sequenzdiagramm anzeigen.

Wenn Sie in einem Sequenzdiagramm Nachrichtenbeziehungen zwischen Objekten erstellen, wird für jede Nachricht eine eigene Beziehungslinie angezeigt. Die Nachrichten in Sequenzdiagrammen verfügen über mehr Eigenschaften, die Sie bearbeiten können, als die Nachrichten in Kollaborationsdiagrammen.

Sie können mit Nachrichten folgende Aktionen ausführen:

1. Eine Selbstbenachrichtigung erstellen.
2. Nachrichtenbeziehungen neu anordnen.
3. Ein Objekt mit einer Nachricht erstellen.
4. Ein Objekt mit einer Nachricht freigeben.
5. Eine Rückgabebeziehung mit Hilfe der Tool-Palette (Toolbox) definieren.
6. Eine Rückgabebeziehung im Objektinspektor (Eigenschaftsfenster) definieren.

So erstellen Sie eine Selbstbenachrichtigung:

1. Klicken Sie in der Tool-Palette auf die Schaltfläche Selbstbenachrichtigung.
2. Wenn Sie mit einem Sequenzdiagramm arbeiten, klicken Sie an der Stelle auf die Lebenslinie des Objekts, an der die Nachricht hinzugefügt werden soll. Wenn Sie auf das Objekt klicken, wird die Selbstbenachrichtigung oben auf der Lebenslinie hinzugefügt. Wenn Sie mit einem Kollaborationsdiagramm arbeiten, klicken Sie auf das Objekt.

So ordnen Sie eine Nachrichtenbeziehung neu an:

1. Öffnen Sie ein Diagramm.
2. Die Reihenfolge der Nachrichten kann auf folgende Arten geändert werden:
 - Ziehen Sie in der Diagrammansicht die Nachrichtenbeziehungen auf der Objektlebenslinie nach oben oder unten. Die Sequenznummern der Beziehungen werden dann automatisch geändert.
 - Ändern Sie im Objektinspektor den Wert des Feldes Sequence Number.
 - Ändern Sie die Sequenznummer in der Diagrammansicht mit dem internen Editor.

So erstellen Sie ein Objekt mit einer Nachricht:

1. Wählen Sie im Sequenzdiagramm eine Nachrichtenbeziehung aus.
2. Klicken Sie im Objektinspektor der Nachrichtenbeziehung auf das Feld Creation.
3. Wählen Sie im Listenfeld **True** aus.

Ergebnis: Die Beziehungslinie zeigt nun auf das Empfängerobjekt und nicht mehr auf dessen Lebenslinie. Das erzeugte Objekt wird auf der Lebenslinie nach unten verschoben, um anzusehen, dass es vom erstellenden Objekt aus gesehen zu einem

späteren Zeitpunkt vorhanden ist.

Die Eigenschaft *Creation* hat im Eigenschaftsfenster standardmäßig den Wert **False**.

So geben Sie ein Objekt mit einer Nachricht frei:

1. Wählen Sie im Sequenzdiagramm eine Nachrichtenbeziehung aus.
2. Klicken Sie im Objektinspektor der Nachrichtenbeziehung auf das Feld *Destruction*.
3. Wählen Sie im Listenfeld **True** aus.

Ergebnis: Das Objekt ist frei gegeben.

Die Eigenschaft *Destruction* hat im Objektinspektor standardmäßig den Wert **False**.

So definieren Sie eine Rückgabebeziehung mit Hilfe der Tool-Palette (Toolbox):

1. Klicken Sie in der Tool-Palette auf die Schaltfläche Rückgabebeziehung.
2. Klicken Sie im Sequenzdiagramm auf die Objektlebenslinie am Anbieterende der Nachrichtenbeziehung, um die Rückgabebeziehung zu erstellen.

So definieren Sie eine Rückgabebeziehung im Objektinspektor (Eigenschaftsfenster):

1. Wählen Sie die Nachrichtenbeziehung im Sequenzdiagramm aus.
2. Klicken Sie im Hauptmenü auf **Ansicht** | Objektinspektor, oder drücken Sie die Taste F4.
3. Klicken Sie im Objektinspektor auf das Feld *Return Arrow*, und wählen Sie den Wert **True** aus.

Siehe auch

Beziehungen umlenken (siehe Seite 130)

UML 1.5-Interaktionsdiagramm (siehe Seite 1302)

1.137 UML 1.5-Zustandsdiagramme erstellen

Folgende Tipps und Techniken helfen Ihnen bei der Arbeit mit UML 1.5-Zustandsdiagrammen.

Gehen Sie zur Erstellung eines UML 1.5-Zustandsdiagramms folgendermaßen vor:

1. Erstellen Sie Start- und Endpunkte.
 2. Erstellen Sie Haupt- und Unterzustände.
- Tip:** Zustände können verschachtelt werden.
3. Erstellen Sie Übergänge.
 4. Erstellen Sie Historienknoten.
 5. Wahlweise können Sie auch Verknüpfungen zu Elementen in anderen Diagrammen erstellen.

So erstellen Sie Eintritts- und Austrittsaktionen:

1. Erstellen Sie im gewünschten Zustand einen internen Übergang.
 2. Doppelklicken Sie auf den Übergang, um den internen Editor zu aktivieren.
 3. Benennen Sie den internen Übergang entsprechend der folgenden Syntax um: Stereotyp/Aktionsname (Argument)
Beispiel: exit/setState(idle)
- Alternativ können Sie einen internen Übergang erstellen und seine Eigenschaften für den Ereignisnamen, die Ereignisargumente und den Aktionsausdruck im Objektinspektor festlegen.

Siehe auch

- [Interne Übergänge erstellen](#) (siehe Seite 201)
- [Verknüpfungen erstellen](#) (siehe Seite 139)
- [UML 1.5-Zustandsdiagramme – Referenz](#) (siehe Seite 1305)

1.138 Pins erstellen

So fügen Sie einen Eingabe-, Ausgabe- oder Wert-Pin hinzu:

1. Klicken Sie mit der rechten Maustaste auf eine Aktion.
2. Wählen Sie **Neu > Eingabe-Pin (oder Ausgabe-Pin bzw. Wert-Pin)**.

Ergebnis: Der Zielaktion wird ein quadratischer Pin hinzugefügt. Beachten Sie, dass die Pins an ihren Aktionen verankert sind und nur entlang des Aktionsrandes verschoben werden können.

Alternative:

1. Öffnen Sie die Tool-Palette.
2. Klicken Sie auf die entsprechende Schaltfläche und danach auf die Zielaktion.

Siehe auch

Pin (siehe Seite 1326)

1.139 UML 2.0-Aktivitätsdiagramme erstellen

Folgende Tipps und Techniken helfen Ihnen bei der Erstellung eines UML 2.0-Aktivitätsdiagramms. Normalerweise werden Aktivitätsdiagramme nach Zustandsmaschinendiagrammen erstellt.

Gehen Sie zur Erstellung eines UML 2.0-Aktivitätsdiagramms folgendermaßen vor:

1. Erstellen Sie eine oder mehrere Aktivitäten. Sie können mehrere Aktivitäten in ein Diagramm einfügen oder für jede Aktivität ein eigenes Diagramm erstellen.

Warnung: Aktivitäten können nicht verschachtelt werden.

2. Normalerweise werden Aktivitäten mit Zuständen oder Übergängen in Zustandsmaschinendiagrammen verbunden. Wechseln Sie in Ihre Zustandsmaschinendiagramme, und ordnen Sie den neu erstellten Aktivitäten Zustände und Übergänge zu.

Tip: Danach stellen Sie möglicherweise fest, dass weitere Aktivitäten erstellt werden müssen, oder dass dieselbe Aktivität an mehreren Positionen verwendet werden kann.

3. Wechseln Sie wieder in das Aktivitätsdiagramm. Überprüfen Sie die Abläufe in den Aktivitäten. Jede Aktivität kann über einen Objektfluss (für die Übertragung von Daten), einen Kontrollfluss und weitere Abläufe verfügen.

4. Erstellen Sie für jeden Ablauf Start- und Endpunkte. Jeder Ablauf kann folgende Startpunkte haben:

- Anfangsknoten
- Aktivitätsparameter (für Objektfluss)
- Aktion "Ereignis akzeptieren"
- Aktion "Zeitereignis akzeptieren" Jeder Ablauf endet mit einem Aktivitätsende- oder Ablaufende-Knoten. Wenn die Aktivität mehrere Startpunkte hat, können diese gleichzeitig verwendet werden.

5. Erstellen Sie Objektknoten. Zwischen Objektknoten und Klassen in Klassendiagrammen werden keine Beziehungen erstellt. Sie können aber Hyperlinks verwenden, um die Interpretation der Diagramme zu erleichtern.

6. Erzeugen Sie Aktionsknoten für die Abläufe. Abläufe können über gemeinsam genutzte Aktionen verfügen.

Warnung: Aktionen können nicht verschachtelt werden.

7. Fügen Sie für Objektflüsse Pins zu Aktionen hinzu. Verbinden Sie Aktionen und Pins über Ablaufbeziehungen.

8. Fügen Sie Vor- und Nachbedingungen hinzu. Sie können Text- oder OCL-Bedingungen erstellen.

9. Wahlweise können Sie auch Verknüpfungen zu Elementen in anderen Diagrammen erstellen.

So fügen Sie einer Aktivität einen Aktivitätsparameter hinzu:

1. Klicken Sie in der Tool-Palette auf die Schaltfläche Aktivitätsparameter.
2. Klicken Sie auf die Zielaktivität. Oder Wählen Sie im Kontextmenü der Aktivität **Hinzufügen > Aktivitätsparameter**.

Ergebnis: Der Aktivität wird ein Aktivitätsparameter-Knoten in Form eines Rechtecks hinzugefügt. Beachten Sie, dass der Aktivitätsparameter mit seiner Aktivität verknüpft ist. Sie können den Knoten nur entlang des Aktivitätsrandes verschieben.

Anmerkung: Aktivitätsparameter können nicht über Kontrollflussbeziehungen verbunden werden.

Siehe auch

Übergang oder Zustand mit einer Aktivität verknüpfen (siehe Seite 186)

Aktionen in einer Aktivität gruppieren (siehe Seite 164)

Verknüpfungen erstellen (siehe Seite 139)

UML 2.0-Aktivitätsdiagramme – Referenz (siehe Seite 1323)

1.140 Aktionen in einer Aktivität gruppieren

Sie können Aktionen auf folgende Arten in einer Aktivität gruppieren:

1. Mit den Schaltflächen der Tool-Palette
2. Per Drag&Drop
3. Über das Kontextmenü des Aktivitätselements

So verwenden Sie die Tool-Palette:

1. Erstellen Sie mit Hilfe der Tool-Palette des Diagramms einen Aktivitätsknoten.
2. Klicken Sie auf die Aktionsschaltfläche und danach auf die Zielaktivität.

So gruppieren Sie Aktionen per Drag&Drop in einer Aktivität:

1. Fügen Sie ein Aktionselement in das Diagramm ein.
2. Ziehen Sie die neue Aktion an den Anfang einer vorhandenen Aktivität, und legen Sie sie dort ab.

So verwenden Sie das Kontextmenü des Aktivitätselements:

1. Klicken Sie mit der rechten Maustaste auf die Zielaktivität.
2. Wählen Sie im Kontextmenü **Hinzufügen ▶ Aktion**.

Siehe auch

UML 2.0-Aktivitätsdiagramme erstellen (siehe Seite 162)

UML 2.0-Aktivitätsdiagramme – Referenz (siehe Seite 1323)

1.141 Mit Objektfluss- und Kontrollflussbeziehungen arbeiten

Sie können einen Kontroll- oder Objektfluss als normale Beziehung zwischen zwei Knotenelementen erstellen. Die gültigen Knoten werden hervorgehoben, wenn Sie die Beziehung einrichten.

Wenn das Zielelement nicht direkt erreichbar ist, können Sie einen Bildlauf durchführen oder das Kontextmenü verwenden.

Es gelten jedoch entsprechend der UML 2.0-Spezifikation bestimmte Einschränkungen:

- Zumindest an einem Ende einer Objektflussbeziehung muss sich ein Objekt befinden.
- Zwei Aktionen können nur über einen Ausgabe-Pin der Quellaktion durch eine Objektflussbeziehung verbunden werden.
- Kontrollflussbeziehungen sind nicht zwischen Objekten und/oder Aktivitätsparametern möglich.

Sie können im Zusammenhang mit Objekt- und Kontrollflüssen folgende Aktionen durchführen:

1. Eine Flussbeziehung erstellen.
2. Eine Aufspaltung oder Zusammenführung erstellen.
3. Eine Entscheidung oder Zusammenfassung erstellen.

So erstellen Sie einen Flussbeziehung:

1. Klicken Sie mit der rechten Maustaste auf das Quellelement.
2. Wählen Sie im Kontextmenü **Hinzufügen**▶**Kontrollablauf** oder **Hinzufügen**▶**Objektfluss**. Das Dialogfeld Ziel auswählen wird geöffnet.
3. Wählen Sie das Ziel aus, und klicken Sie auf **OK**. Beachten Sie, dass die Schaltfläche **OK** nur aktiv ist, wenn Sie ein gültiges Ziel ausgewählt haben.

So erstellen Sie eine Aufspaltung oder Zusammenführung:

1. Überlegen Sie sich, welche Aktionen beteiligt sein sollen. Fügen Sie ggf. zuerst alle Aktionen in das Diagramm ein, und positionieren Sie sie wie gewünscht.
2. Fügen Sie eine Aufspaltung oder Zusammenführung in das Diagramm ein. Nehmen Sie die entsprechenden Größenänderungen vor.
3. Wenn Sie mehrere Quellaktionen verwenden möchten, erstellen Sie eine Kontrollflussbeziehung von jeder Aktion zur Zusammenführung und von dieser eine Kontrollflussbeziehung zur Zielaktion. Wenn Sie mehrere Zielaktionen verwenden möchten, erstellen Sie eine Kontrollflussbeziehung von der Quellaktion zur Aufspaltung und von dieser eine Kontrollflussbeziehung zu jeder Zielaktion.

So erstellen Sie eine Entscheidung oder Zusammenfassung:

1. Überlegen Sie sich, welche Aktionen beteiligt sein sollen. Fügen Sie ggf. zuerst alle Aktionen in das Diagramm ein, und positionieren Sie sie wie gewünscht.
2. Fügen Sie eine Entscheidung oder Zusammenfassung in das Diagramm ein. Nehmen Sie die entsprechenden Größenänderungen vor.
3. Wenn Sie mehrere Aktionen zusammenfassen möchten, erstellen Sie eine Kontrollflussbeziehung von jeder Quellaktion zur Zusammenfassung und von dieser eine Kontrollflussbeziehung zur Zielaktion. Wenn Sie eine Entscheidung implementieren möchten, erstellen Sie eine Kontrollflussbeziehung von der Quellaktion zur Entscheidung und von dieser eine Kontrollflussbeziehung zu jeder Zielaktion.

Siehe auch

Eine einfache Beziehung erstellen ( siehe Seite 141)

UML 2.0-Aktivitätsdiagramm ( siehe Seite 1323)

1.142 UML 2.0-Komponentendiagramme erstellen

Folgende Tipps und Techniken helfen Ihnen bei der Arbeit mit UML 2.0-Komponentendiagrammen. Es kann hilfreich sein, die Erstellung eines Modells mit Komponentendiagrammen zu beginnen. Dies gilt besonders bei der Modellierung eines großen Systems, beispielsweise eines verteilten Client-Server-Softwaresystems mit zahlreichen verknüpften Modulen. Verwenden Sie Komponentendiagramme zur Modellierung der logischen Struktur und Verteilungsdiagramme zur Modellierung der physischen Struktur des Systems.

Gehen Sie zur Erstellung eines UML 2.0-Komponentendiagramms folgendermaßen vor:

1. Erstellen Sie eine Hierarchie von Komponenten. Die größte Komponente kann das gesamte System oder dessen Hauptbestandteil sein (beispielsweise *Server-Anwendung*, *IDE* oder *Service*).

Tip: Komponentenknoten können verschachtelt werden. Es gibt zwei Verfahren für die Erstellung eines verschachtelten Komponentenknotens: Sie können eine vorhandene Komponente auswählen und eine untergeordnete Komponente einfügen. Alternativ können Sie zwei separate Komponenten erstellen und sie über eine Assoziations-Kompositions-Beziehung miteinander verbinden.

2. Sie können die Komponentenhierarchie abschließen, indem Sie konkrete Klassen und Instanzspezifikationen hinzufügen. Diese lassen sich direkt in einem Komponentendiagramm erstellen. Sie können sie aber auch in einem Klassendiagramm erzeugen und dann Verknüpfungen in einem Komponentendiagramm einrichten.
3. Erstellen Sie Interfaces. Jede Komponente kann über ein bereitgestelltes und ein erforderliches Interface verfügen.
4. Erstellen Sie gegebenenfalls Artefakte. Normalerweise werden physische Systemartefakte in Verteilungsdiagrammen beschrieben. Wenn jedoch Komponenten eng mit ihrem physischen Speicherort verknüpft sind, fügen Sie ein Artefakt in ein Komponentendiagramm ein und erstellen eine Beziehung.

Tip: Artefakte können verschachtelt werden.

5. Erstellen Sie gegebenenfalls Ports für die Komponenten. Sie können einer Komponente einen Port zuweisen und ihn mit verschiedenen Klassen oder Komponenten verbinden. In diesem Fall entscheidet der Port bei Eingang einer Nachricht, von welcher Klasse sie bearbeitet wird.
6. Erzeugen Sie Beziehungen zwischen Elementen.
7. Wahlweise können Sie auch Verknüpfungen zu Elementen in anderen Diagrammen erstellen.

Siehe auch

Mit bereitgestellten und erforderlichen Interfaces arbeiten ([siehe Seite 207](#))

Verknüpfungen erstellen ([siehe Seite 139](#))

UML 2.0-Komponentendiagramme – Referenz ([siehe Seite 1326](#))

1.143 Delegationskonnektoren erstellen

So erstellen Sie einen Delegationskonnektor:

1. Klicken Sie mit der rechten Maustaste auf ein Interface, und wählen Sie im Kontextmenü **Neu > Delegationskonnektor**.
2. Wählen Sie im Dialogfeld Ziel auswählen das Ziel-Interface im Modell oder in den Favoriten aus.
3. Klicken Sie auf OK.

Siehe auch

UML 2.0-Kompositionssstrukturdiagramm (siehe Seite 1329)

1.144 Eine interne Struktur für einen Knoten erzeugen

So erstellen Sie eine interne Struktur für einen Knoten:

1. Klicken Sie in der Tool-Palette des Diagramms auf die Schaltfläche **Part**.
2. Klicken Sie auf einen gültigen Container (Klasse oder Kollaboration).
3. Wiederholen Sie diese Schritte für alle erforderlichen Teilnehmer.

Tip: Sie können bei gedrückter STRG-Taste in der Tool-Palette des Diagramms auf die Schaltfläche **Part** klicken. Bei jedem Klick auf einen gültigen Container wird ein neues Part erstellt.

4. Verbinden Sie die Kollaborationsparts über Konnektoren.
5. Legen Sie im Objektinspektor die Eigenschaften des Parts fest.

Siehe auch

UML 2.0-Kompositionssstrukturdiagramm (siehe Seite 1329)

1.145 Ports erstellen

So erstellen Sie einen Port:

1. Klicken Sie in der Tool-Palette des Diagramms auf die Schaltfläche **Port**.
2. Klicken Sie auf die Zielklasse oder das Ziel-Part.
3. Erstellen Sie alle erforderlichen Ports.

Siehe auch

UML 2.0-Kompositionsstrukturdiagramm (siehe Seite 1329)

1.146 Ein referenziertes Part erstellen

So erstellen Sie ein referenziertes Part:

1. Öffnen Sie die Diagrammansicht.
2. Verwenden Sie eines der folgenden Verfahren:
 - Klicken Sie in der Tool-Palette des Diagramms auf die Schaltfläche **Referenziertes Part**.
 - Klicken Sie mit der rechten Maustaste auf einen Zielcontainer, und wählen Sie im Kontextmenü **Neu > Referenziertes Part**.
 - Wählen Sie ein Part, öffnen Sie die Modellansicht, und aktivieren Sie die Option **Aggregation mittels Referenz**.

Siehe auch

UML 2.0-Kompositionssstrukturdiagramm (siehe Seite 1329)

1.147 Mit Kollaborationsverwendungen arbeiten

So erstellen Sie eine Kollaborationsverwendung:

1. Klicken Sie in der Tool-Palette des Diagramms auf die Schaltfläche **Kollaborationsverwendung**.
2. Klicken Sie auf den Zielcontainer.
3. Geben Sie den Namen der Kollaborationsverwendung ein.

So erstellen Sie eine Beziehung zu einem Kollaborationstyp:

1. Wählen Sie ein Kollaborationsverwendungselement aus.
2. Legen Sie den Typ der Kollaborationsverwendung auf eine der folgenden Arten fest:
 - Klicken Sie im Objektinspektor im Feld **Type** der Kollaborationsverwendung auf die Auswahlfläche, und wählen Sie die zu instantierende Kollaboration im Modell oder in den Favoriten aus.
 - Geben Sie nach dem Namen der Kollaborationsverwendung einen Doppelpunkt und danach den Namen der Kollaboration ein, die instantiiert werden soll.

Ergebnis: Der Typ der Kollaborationsverwendung wird neben ihrem Namen angezeigt.

So lösen Sie die Beziehung zu einem Kollaborationstyp:

1. Klicken Sie mit der rechten Maustaste auf die Kollaborationsverwendung, der ein bestimmter Typ zugeordnet ist.
2. Klicken Sie im Kontextmenü auf **Kollaborationsbeziehung lösen**.

So binden Sie eine Kollaborationsverwendung an eine Rolle (Part):

1. Klicken Sie in der Tool-Palette auf die Schaltfläche **Rollenbindung**.
2. Wenn Sie mit der Maus auf die Client-Kollaborationsverwendung zeigen, wird der gültige Client durch eine schwarze Ellipse hervorgehoben.
3. Ziehen Sie die Rollenbindungsbeziehung auf das Anbieter-Part. Wenn das Zielelement gültig ist, wird es hervorgehoben.
4. Geben Sie den Rollennamen ein, und drücken Sie die **EINGABETASTE**, um den internen Editor zu schließen.

Wenn eine Kollaborationsverwendung einer Kollaboration zugeordnet ist, die Parts (Rollen) enthält, können Sie diese an die Parts (Rollen) eines anderen Klassifizierers binden.

Führen Sie dann die Bindung der Parts (Rollen) der verschiedenen Klassifizierer über die Kollaborationsverwendung wie folgt durch:

1. Erstellen Sie eine Kollaborationsverwendung, und definieren Sie ihren Typ.
2. Erstellen Sie ein oder mehrere Parts in der Kollaboration.
3. Klicken Sie mit der rechten Maustaste auf die Kollaborationsverwendung, und wählen Sie **Neue Rolle binden**.
4. Wählen Sie im Dialogfeld **Ziel auswählen** die Rolle aus, die an den Zielklassifizierer gebunden werden soll.

Ergebnis: Eine Rollenbeziehung wird von der Kollaborationsverwendung zur Rolle im Zielklassifizierer erstellt. Die Rollenbeziehung wird mit dem Namen der in der Kollaboration ausgewählten Rolle angezeigt.

Anmerkung: Jede Rolle kann nur einmal gebunden werden. Wenn Sie den Befehl **Neue Rolle binden** das nächste Mal aufrufen, wird die zuvor ausgewählte Rolle nicht mehr angezeigt.

So erstellen Sie einen Eigentümer:

1. Klicken Sie mit der rechten Maustaste auf eine Kollaborationsverwendung, und wählen Sie im Kontextmenü Objektinspektor.
2. Klicken Sie im Objektinspektor im Feld **Owning classifier** auf die Auswahlsschaltfläche.
3. Wählen Sie im Dialogfeld Besitzenden Klassifizierer auswählen die Eigentümerklasse oder -kollaboration aus, und klicken Sie auf **OK**.

Ergebnis: Zwischen dem Eigentümer (Anbieter) und der Kollaborationsverwendung (Client) wird eine Beziehung erstellt. Die Beziehung ist mit <<represents>> beschriftet.

Siehe auch

UML 2.0-Kompositionssstrukturdiagramm (siehe Seite 1329)

1.148 UML 2.0-Verteilungsdiagramme erstellen

Folgende Tipps und Techniken helfen Ihnen bei der Erstellung eines UML 2.0-Verteilungsdiagramms. Es kann hilfreich sein, die Erstellung eines Modells mit Verteilungsdiagrammen zu beginnen. Dies gilt besonders bei der Modellierung eines großen Systems mit zahlreichen Modulen, die sich auf verschiedenen Computern befinden. Verwenden Sie Verteilungsdiagramme zur Modellierung der physischen Struktur und Komponentendiagramme zur Modellierung der logischen Struktur des Systems.

Gehen Sie zur Erstellung eines UML 2.0-Verteilungsdiagramms folgendermaßen vor:

1. Erstellen Sie eine Hierarchie von Ausführungsumgebungen, Geräten und Knoten. Ausführungsumgebungen repräsentieren in der Regel die Software-Umgebung für die Ausführung des Systems (beispielsweise ein Betriebssystem). Geräte stellen normalerweise die Hardware-Ausstattung dar, etwa einen Drucker, ein Festplattlaufwerk oder einen Computer. Knoten sind der Rest der physischen Entitäten, beispielsweise Dateien.

Tip: Ausführungsumgebungen, Geräte und Knoten können verschachtelt werden. Beispielsweise können Sie einen Knoten in eine Ausführungsumgebung oder in ein Gerät einfügen.

2. Erstellen Sie Artefakte.
3. Erstellen Sie Verteilungs- und Instanzspezifikationen. Hierbei legen Sie die physischen Positionen von Objekten und anderen Entitäten des Systems fest.
4. Fügen Sie Artefakten Operationen hinzu.
5. Anschließend können Sie im Objektinspektor die Eigenschaften der Operation (Parameter, Stereotyp, Multiplizität usw.) festlegen.
6. Wahlweise können Sie auch Verknüpfungen zu Elementen in anderen Diagrammen erstellen.

So verteilen Sie ein Artefakt an einen Zielknoten:

1. Klicken Sie in der Tool-Palette des Diagramms auf die Schaltfläche **Verteilung**.
2. Klicken Sie auf das gewünschte Artefakt. Die gültige Quelle wird durch einen durchgezogenen Rahmen markiert.
3. Ziehen Sie die Verteilungsbeziehung zum gewünschten Zielknoten, und lassen Sie die Maustaste los. Der gültige Zielknoten wird durch einen durchgezogenen Rahmen hervorgehoben.

So definieren Sie die Parameter einer Operation:

1. Wählen Sie die gewünschte Operation in einem Artefakt aus.
2. Erweitern Sie im Objektinspektor den Knoten Allgemein, und wählen Sie das Feld Parameters aus.
3. Klicken Sie auf die Auswahlabschaltfläche, um das Dialogfeld Parameter hinzufügen/entfernen zu öffnen.
4. Klicken Sie auf Hinzufügen. In der Parameterliste wird nun ein neuer Eintrag erstellt.
5. Geben Sie den Namen, die Typmultiplizität, den Standardwert und die Richtung des Parameters an. Der Parametertyp kann in der Liste der vordefinierten Typen oder im Modell ausgewählt werden.
6. Stellen Sie die Parameterliste mit den Schaltflächen Hinzufügen und Entfernen zusammen.
7. Klicken Sie anschließend auf OK.

Siehe auch

Verknüpfungen erstellen (siehe Seite 139)

UML 2.0-Verteilungsdiagramme – Referenz (siehe Seite 1328)

1.149 Lebenslinien mit Klassifizierern verknüpfen

So ordnen Sie einer Lebenslinie einen Klassifizierer zu:

1. Wählen Sie eine Lebenslinie in einem Interaktionsdiagramm aus.
2. Klicken Sie mit der rechten Maustaste, und wählen Sie **Auswählen > Typ**. Das Dialogfeld Typ des repräsentierten verbindbaren Elements auswählen wird geöffnet.
3. Wählen Sie in der Baumstruktur der verfügbaren Modellelementen den Klassifizierer aus, der der Lebenslinie zugeordnet werden soll.
4. Klicken Sie auf OK.

Siehe auch

Klassifizierer instantiiieren (siehe Seite 150)

UML 2.0-Interaktionsdiagramme (siehe Seite 1315)

1.150 Ausführungs- und Aufrufspezifikationen kopieren und einfügen

Mit Ausführungs- und Aufrufspezifikationen können Zwischenablageoperationen durchgeführt werden.

So können Sie eine Ausführungs- oder Aufrufspezifikation kopieren und einfügen:

1. In den Kontextmenüs der Spezifikationen stehen die Befehle Ausschneiden, Kopieren und Einfügen zur Verfügung. Die Elemente können im selben oder in ein anderes Diagramm kopiert und verschoben werden.
2. Wenn Sie eine Spezifikation kopieren, wird der gesamte Nachrichtenzweig kopiert. Beim Einfügen eines Elements in eine Lebenslinie wird die Sequenznummer entsprechend der vorhandenen Nachrichtennummern geändert.
3. Wenn Sie eine Ausführungs- oder Aufrufspezifikation in ein anderes Diagramm einfügen, werden auch alle Nachrichten zusammen mit den zugehörigen Lebenslinien eingefügt. Falls das Zieldiagramm keine Lebenslinien für die Ausführungsspezifikation enthält, werden sie automatisch erstellt.

Tip: Sie können Nachrichtenzweige auch per Drag&Drop verschieben und kopieren. Um eine Ausführungs- oder Aufrufspezifikation zu verschieben, ziehen Sie sie einfach an die gewünschte Position. Wenn Sie eine Kopie erstellen möchten, halten Sie während des Ziehens die Taste **STRG** gedrückt.

Siehe auch

Mit UML 2.0-Nachrichten arbeiten (siehe Seite 182)

UML 2.0-Interaktionsdiagramme (siehe Seite 1315)

1.151 Sequenz- oder Kommunikationsdiagramm aus einer Interaktion erstellen

So erstellen Sie ein Sequenz- oder Kommunikationsdiagramm aus einer Interaktion:

1. Wählen Sie in der Modellansicht ein Interaktionselement aus.
2. Klicken Sie mit der rechten Maustaste auf den Knoten **Interaktion**, und wählen Sie **Mit Sequenzdiagramm 2.0 öffnen** oder **Mit Kommunikationsdiagramm 2.0 öffnen**.

Ergebnis: Wenn das Diagramm noch nicht vorhanden ist, wird es erstellt. Anschließend wird es in der Diagrammansicht geöffnet.

Siehe auch

UML 2.0-Interaktionsdiagramme (siehe Seite 1315)

1.152 Zustandsinvarianten erstellen

So erstellen Sie eine Zustandsinvariante als OCL-Ausdruck:

1. Klicken Sie in der Tool-Palette des Diagramms auf die Schaltfläche **Zustandsinvariante**.
2. Klicken Sie auf das gewünschte Ziellelement (Lebenslinie oder Ausführungsspezifikation).

Tip: Verwenden Sie alternativ im Kontextmenü einer Lebenslinie oder Ausführungsspezifikation den Befehl **Hinzufügen > Zustandsinvariante**.

3. Erweitern Sie im Objektinspektor der Zustandsinvariante den Knoten **Allgemein**.
4. Wählen Sie in der Dropdown-Liste des Feldes **Invariant kind** den Eintrag **OCL expression** aus. Die Form des Diagrammelements wird nun in ein Paar geschweifte Klammern geändert.
5. Wählen Sie im Knoten **OCL-Invariante**, der nun zusätzlich im Eigenschaftsfenster angezeigt wird, die Sprache des Kommentars in der Dropdown-Liste **Sprache** aus. Die möglichen Optionen sind **OCL** und **Text**.
6. Geben Sie den Text ein, und übernehmen Sie die Änderungen.

So verbinden Sie eine Zustandsinvariante mit einem Zustand:

1. Klicken Sie in der Tool-Palette des Diagramms auf die Schaltfläche **Zustandsinvariante**.
2. Klicken Sie auf das gewünschte Ziellelement (Lebenslinie oder Ausführungsspezifikation).
3. Erweitern Sie im Objektinspektor der Zustandsinvariante den Knoten **Allgemein**.
4. Wählen Sie in der Dropdown-Liste des Feldes **Invariant kind** den Eintrag **States/Regions** aus.
5. Klicken Sie im Feld **States/Regions** auf die Auswahlfläche.
6. Wählen Sie im Dialogfeld Zustände und/oder Regionen auswählen die gewünschten Zustände und/oder Regionen im Modell aus, und klicken Sie auf die Schaltfläche **Hinzufügen**.
7. Klicken Sie dann auf **OK**.

Tip: Geben Sie alternativ den Zustands- oder Regionsnamen ein. Wenn der Zustand oder die Region zu einem anderen Paket gehört, geben Sie den vollständigen Namen ein.

Siehe auch

Überblick zur OCL-Unterstützung (siehe Seite 1453)

UML 2.0-Interaktionsdiagramme (siehe Seite 1315)

1.153 UML 2.0-Sequenzdiagramme oder -Kommunikationsdiagramme erstellen

Folgende Tipps und Techniken helfen Ihnen bei der Erstellung eines UML 2.0-Sequenz- oder -Kommunikationsdiagramms. Normalerweise werden Interaktionsdiagramme nach Klassendiagrammen erstellt.

Beim Erstellen eines Interaktionsdiagramms wird die entsprechende Interaktion dem Projekt hinzugefügt. Interaktionen werden in der Modellansicht als Knoten angezeigt.

Anmerkung: Die Darstellung einer Interaktion in der Modellansicht richtet sich nach dem Ansichtstyp, den Sie in den Modellansicht-Optionen auf Vorgabe- oder Projektgruppen-Ebene festgelegt haben. Wenn der Modus *Modellzentrisch* ausgewählt ist, wird eine Interaktion unter ihrem Paketknoten und Diagrammknoten angezeigt. Im Modus *Diagrammzentrisch* wird sie lediglich unter ihrem Diagrammknoten angezeigt.

Anmerkung: Sie können eine Interaktion auf zwei Arten anzeigen: als Sequenzdiagramm oder als Kommunikationsdiagramm. Alle mit der einen Ansicht durchgeführten Aktionen werden automatisch in die andere Ansicht übernommen. Wenn Sie Elemente in einer Interaktion hinzufügen oder entfernen, wird diese Änderung auch im zugehörigen Interaktionsdiagramm durchgeführt (und umgekehrt). Jedes Interaktionsdiagramm enthält einen Verweis auf die zugrunde liegende Interaktion.

Anmerkung: Im Gegensatz zu UML 1.5 können vorhandene Diagramme nicht zwischen Sequenz- und Kommunikationsdiagramm umgewandelt werden. Sie können jedoch ein Sequenzdiagramm und ein Kommunikationsdiagramm anhand derselben Interaktion erstellen.

Gehen Sie zur Erstellung eines UML 2.0-Sequenzdiagramms folgendermaßen vor:

1. Erstellen Sie eine Interaktionsverwendung.
2. Navigieren Sie zu einer referenzierten Interaktion.
3. Ordnen Sie einer Lebenslinie ein referenziertes Element zu.
4. Ordnen Sie einer Lebenslinie einen Typ zu.
5. Definieren Sie eine Dekomposition für eine Lebenslinie.
6. Wiederholen Sie diese Schritte für alle erforderlichen Interaktionen.
7. Stellen Sie mit Hilfe von Nachrichten Beziehungen zwischen den erzeugten Lebenslinien her.

So erstellen Sie eine Interaktionsverwendung:

1. Klicken Sie in der Tool-Palette des Diagramms auf die Schaltfläche Interaktionsverwendung.
2. Klicken Sie auf die Ziellebenslinie.

Tip: Verwenden Sie alternativ in der Diagrammansicht oder der Modellansicht den Befehl Hinzufügen im Kontextmenü der Lebenslinie.

3. Aktivieren Sie im Objektinspektor die Registerkarte **Eigenschaften** der neuen Interaktionsverwendung.
4. Klicken Sie im Feld **Interaction name** auf die Auswahlshaltfläche.

Tip: Geben Sie alternativ den Namen der Interaktion ein.

5. Wählen Sie im Dialogfeld Referenzierte Interaktion auswählen die gewünschte Interaktion im Projekt oder in den Favoriten

aus, und klicken Sie auf OK.

Neue Interaktionsverwendungen sind anfangs an der Lebenslinie verankert. Sie können auf mehrere Lebenslinien erweitert, von Lebenslinien gelöst und wieder an ihnen verankert werden.

So gelangen Sie zu einer referenzierten Interaktion:

1. Klicken Sie mit der rechten Maustaste auf eine Interaktionsverwendung, die auf eine andere Interaktion zeigt.
2. Klicken Sie im Kontextmenü auf Auswählen.
3. Wählen Sie im Untermenü das gewünschte Ziel aus.

So ordnen Sie einer Lebenslinie ein referenziertes Element zu:

1. Vergewissern Sie sich, dass die Elemente, die Sie Lebenslinien zuordnen möchten, im Projekt vorhanden sind.
2. Wählen Sie in der Modellansicht oder Diagrammansicht die gewünschte Lebenslinie aus.
3. Klicken Sie im Objektinspektor auf das Feld **Represents**.
4. Klicken Sie auf die Auswahlorschaltfläche.
5. Wählen Sie im Dialogfeld Repräsentiertes verbindbares Element auswählen das gewünschte Element im Projekt oder in den Favoriten aus.
6. Klicken Sie auf OK.

So ordnen Sie einer Lebenslinie einen Typ zu:

1. Wählen Sie in der Modellansicht oder Diagrammansicht die gewünschte Lebenslinie aus.
2. Klicken Sie im Objektinspektor auf das Feld **Type**.
3. Klicken Sie auf die Auswahlorschaltfläche.
4. Wählen Sie im Dialogfeld Typ des repräsentierten verbindbaren Elements auswählen die Klasse mit der Definition des gewünschten Typs im Projekt oder in den Favoriten aus.
5. Klicken Sie auf OK.

So definieren Sie eine Dekomposition für eine Lebenslinie:

1. Wählen Sie in der Modellansicht oder Diagrammansicht die gewünschte Lebenslinie aus.
2. Klicken Sie im Objektinspektor auf das Feld **Decomposition**.
3. Klicken Sie auf die Auswahlorschaltfläche.
4. Wählen Sie im Dialogfeld Referenzierte Interaktion auswählen die gewünschte Interaktion im Projekt oder in den Favoriten aus.
5. Klicken Sie auf OK.

Tip: Die Änderungen der oben genannten Eigenschaften werden auch in das entsprechende Kommunikationsdiagramm übernommen.

Siehe auch

UML 2.0-Interaktionsdiagramme – Referenz (siehe Seite 1315)

1.154 Verknüpfung zu einer Interaktion in ein Interaktionsdiagramm einfügen

So fügen Sie eine Verknüpfung zu einer Interaktion in ein Interaktionsdiagramm ein:

1. Öffnen Sie ein Interaktionsdiagramm.
2. Klicken Sie mit der rechten Maustaste auf den Diagrammhintergrund, und wählen Sie **Hinzufügen**▶**Verknüpfung**.
3. Fügen Sie eine Verknüpfung zu einer anderen Interaktion im Projekt hinzu.

Siehe auch

UML 2.0-Interaktionsdiagramme (↗ siehe Seite 1315)

1.155 Mit UML 2.0-Nachrichten arbeiten

In diesem Abschnitt erfahren Sie, wie Sie in Sequenz- und Kommunikationsdiagrammen mit Nachrichten arbeiten können. Obwohl diese beiden Diagrammtypen äquivalent sind, unterscheiden sie sich doch hinsichtlich der Verwendung von Nachrichten.

Sie können mit UML 2.0-Nachrichten folgende Aktionen durchführen:

1. Die Antwortnachricht ein- und ausblenden.
2. Nachrichten verschachteln.
3. Eine Nachricht von einer Lebenslinie zurück zu sich selbst erstellen.
4. Eine Nachrichtenbeziehung für einen Operationsaufruf erstellen.

So blenden Sie die Antwortnachricht ein oder aus:

1. Klicken Sie in einem Interaktionsdiagramm mit der rechten Maustaste auf eine Aufrufnachricht.
2. Aktivieren oder deaktivieren Sie auf der Registerkarte **Beziehung** des Objektinspektors die Option **Show reply message**.

So verschachteln Sie Nachrichten:

1. Sie können Nachrichten verschachteln, indem Sie eine Ausführungsspezifikation als Ursprung der Nachrichtenbeziehungen verwenden. Bei den verschachtelten Nachrichten wird die Nummerierung der übergeordneten Nachricht weitergeführt. Wenn beispielsweise die übergeordnete Nachricht die Nummer 1 hat, erhält ihre erste verschachtelte Nachricht die Nummer 1.1.
2. Sie können auch Nachrichtenbeziehungen zurück zu den übergeordneten Ausführungsspezifikationen erstellen.

So erstellen Sie eine Nachricht von einer Lebenslinie zurück zu sich selbst:

1. Klicken Sie in der Tool-Palette auf die Schaltfläche Nachricht.
2. Klicken Sie in einem Sequenzdiagramm zweimal an der Stelle auf die Lebenslinie, an der die Nachricht eingefügt werden soll. In einem Kommunikationsdiagramm können Sie an einer beliebigen Stelle zweimal auf die Lebenslinie klicken.

So erstellen Sie eine Nachrichtenbeziehung für einen Operationsaufruf:

1. Erstellen Sie eine Interaktion.
2. Erstellen Sie eine Nachrichtenbeziehung zwischen zwei Lebenslinien in der Interaktion.
3. Öffnen Sie im Objektinspektor die Registerkarte Beziehung der Nachrichtenbeziehung.
4. Klicken Sie im Feld **Signature** auf die Schaltfläche **Durchsuchen**.
5. Wählen Sie im Modell oder in den Favoriten die gewünschte Operation aus.
6. Klicken Sie auf OK.

Die neue Nachrichtenbeziehung wird entsprechend der Operation benannt.

Siehe auch

Mit Instanzspezifikationen arbeiten (siehe Seite 205)

UML 2.0-Interaktionsdiagramm (siehe Seite 1315)

UML 2.0-Nachricht (siehe Seite 1318)

Ausführungs- und Aufrufspezifikation (siehe Seite 1320)

1.156 Mit kombinierten Fragmenten arbeiten

So erstellen Sie ein kombiniertes Fragment:

1. Klicken Sie in der Tool-Palette des Diagramms auf die Schaltfläche **Kombiniertes Fragment** und danach auf die Ziellebenslinie.
2. Wählen Sie im Dialogfeld Typ wählen den gewünschten Operator in der Liste der verfügbaren Operatoren aus.

Sie können ein kombiniertes Fragment auch über das Kontextmenü der Modellansicht oder Diagrammansicht erstellen.

Wählen Sie in der Modellansicht oder Diagrammansicht die gewünschte Lebenslinie oder Ausführungsspezifikation aus. Klicken Sie mit der rechten Maustaste, und wählen Sie **Hinzufügen > Kombiniertes Fragment**. Dem ausgewählten Zielelement wird nun ein kombiniertes Fragment hinzugefügt.

Ergebnis: Das kombinierte Fragment wird der Lebenslinie oder Ausführungsspezifikation hinzugefügt. Jedes neue Fragment hat eine andere Farbe, damit es von den anderen kombinierten Fragmenten in derselben Gruppe verschachtelter Frames unterschieden werden kann.

So erstellen Sie einen verschachtelten Operator:

1. Wählen Sie das gewünschte kombinierte Fragment aus.
2. Klicken Sie im Objektinspektor im Feld **Operatoren** auf die Auswahlfläche. Das Dialogfeld Operatoren von kombinierten Fragmenten bearbeiten wird geöffnet.
3. Wählen Sie im Kombinationsfeld **Operator bearbeiten** den gewünschten Operator aus. Wenn für einen bestimmten Operator Parameter benötigt werden, geben Sie die Parameterwerte in das benachbarte Feld ein. Trennen Sie die Werte jeweils durch ein Komma.
4. Klicken Sie auf die Schaltfläche Hinzufügen. Unter dem vorhandenen Eintrag wird eine neue Zeile in der Operatorliste und im Deskriptor des kombinierten Fragments eingefügt.
5. Stellen Sie die gewünschte Operatorliste mit den Schaltflächen Hinzufügen und Entfernen zusammen. Mit den Schaltflächen Auf und Ab können Sie die Reihenfolge der Einträge ändern.
6. Klicken Sie auf Fertig, um die Änderungen zu übernehmen.

Ergebnis: Die verschachtelten Operatoren werden in der festgelegten Reihenfolge im Deskriptor des kombinierten Fragments angezeigt.

Sie können verschachtelte kombinierte Fragmente erstellen, indem Sie einen neuen Fragmentknoten in einem vorhandenen Knoten einfügen. Der neue Knoten wird dann in einer anderen Farbe angezeigt. Die verwendeten Farben sind zufällig. Sie können mit den inneren Frames auf dieselbe Weise wie mit den äußeren Frames arbeiten: sie auf einer Lebenslinie verschieben, auf mehrere Lebenslinien erweitern, von Lebenslinien lösen und wieder an ihnen verankern. Wenn Sie als Quellelement einer Nachrichtenbeziehung einen Frame verwenden, wird dieser automatisch zusammen mit seinen äußeren Frames (falls vorhanden) erweitert.

So erstellen Sie einen Operanden:

1. Wählen Sie in der Modellansicht oder Diagrammansicht das gewünschte kombinierte Fragment aus.
2. Klicken Sie mit der rechten Maustaste, und wählen Sie **Hinzufügen > Interaktionsoperand**.
3. Wählen Sie im Knoten **Interaktionseinschränkung** die Sprache für die Einschränkung aus. Klicken Sie dazu in der Dropdown-Liste **Sprache** auf den gewünschten Eintrag (OCL oder Text).
4. Geben Sie den Einschränkungsausdruck ein.
5. Fügen Sie bei Bedarf noch weitere Operanden hinzu.

6. Übernehmen Sie die Änderungen.

Ergebnis: Ein neuer Operand wird erstellt. Der Einschränkungstext wird im Operandenabschnitt des kombinierten Fragments angezeigt.

Siehe auch

Überblick zur OCL-Unterstützung ( siehe Seite 1453)

UML 2.0-Interaktionsdiagramm ( siehe Seite 1315)

1.157 Frames verankern

So erweitern Sie einen Frame auf mehrere Lebenslinien:

1. Klicken Sie in der Tool-Palette des Diagramms auf die Schaltfläche **Frame verankern**.
2. Klicken Sie auf das Element (kombiniertes Fragment oder Interaktion), das Sie erweitern möchten.
3. Ziehen Sie das Element zur Ziellebenslinie, und lassen Sie die Maustaste los.

Ergebnis: Der Frame wird bis zur Ziellebenslinie erweitert und mit einem Punkt an dieser verankert.

Siehe auch

UML 2.0-Interaktionsdiagramme (↗ siehe Seite 1315)

1.158 Übergang oder Zustand mit einer Aktivität verknüpfen

Sie können eine Aktivität in einem UML 2.0-Aktivitätsdiagramm mit einem Zustand (beim Eintritt in den Zustand, während der Ausführung der Aktivität oder beim Austritt aus dem Zustand) oder mit einem Übergang zwischen Zuständen verknüpfen.

So ordnen Sie einem Zustand eine Aktivität zu:

1. Wählen Sie in einem UML 2.0-Zustandsmaschinendiagramm einen Übergang oder einen Zustand aus.
2. Klicken Sie unter dem Knoten **Allgemein** des Objektinspektors auf das Feld Effect (für einen Übergang) bzw. auf das Feld Do activity, Entry oder Exit (für einen Zustand).
3. Klicken Sie auf die Auswahlschaltfläche, um das Dialogfeld Aktivität auswählen zu öffnen.
4. Wählen Sie die gewünschte Aktivität in der Baumstruktur des Modells aus.
5. Klicken Sie auf OK.

Tip: Nachdem Sie eine Überwachungsbedingung oder einen Effekt im Objektinspektor definiert haben, können Sie diese Werte im Diagramm bearbeiten. Doppelklicken Sie dazu auf den Ausdruck, um den internen Editor zu öffnen.

Siehe auch

Überwachungsbedingung für einen Übergang erstellen (siehe Seite 187)

UML 2.0-Zustandsmaschinendiagramme – Referenz (siehe Seite 1321)

Zustand (siehe Seite 1307)

1.159 Überwachungsbedingung für einen Übergang erstellen

So erstellen Sie eine Überwachungsbedingung für einen Übergang:

1. Wählen Sie im Diagramm einen Übergang aus.
2. Klicken Sie im Knoten **Allgemein** des Objektinspektors auf das Feld Guard.
3. Geben Sie den Bedingungsausdruck ein, und übernehmen Sie die Änderungen.

Siehe auch

Überblick zur OCL-Unterstützung (OCL-Ausdruck) (siehe Seite 1453)

UML 2.0-Zustandsmaschinendiagramme (siehe Seite 1321)

1.160 Historienelement erstellen

So erstellen Sie ein Historienelement für einen Zustand:

1. Wählen Sie in einem Zustand die Region aus, der Sie die Historie hinzufügen möchten.
2. Klicken Sie in der Tool-Palette des Diagramms auf Flache Historie oder Tiefe Historie.
3. Klicken Sie auf die Zielregion.

Siehe auch

Historie (Zustandsmaschinendiagramme) (siehe Seite 1323)

UML 2.0-Zustandsmaschinendiagramme (siehe Seite 1321)

1.161 Member für einen Zustand erzeugen

So erstellen Sie ein Member für einen Zustand:

1. Öffnen Sie die Diagrammansicht.
2. Klicken Sie mit der rechten Maustaste auf einen Zustand, und wählen Sie im Kontextmenü **Hinzufügen > (Member)**. Die folgenden Member stehen zur Wahl:
 - Interne Transaktion
 - Eintrittspunkt
 - Austrittspunkt
 - Region

Siehe auch

UML 2.0-Zustandsmaschinendiagramm (siehe Seite 1321)

1.162 Zustände erstellen

So erstellen Sie einen Zustand:

- Über die Schaltflächen in der Tool-Palette: Klicken Sie in der Tool-Palette des Diagramms auf die Schaltfläche **Zustand**. Klicken Sie im Diagramm auf die Stelle, an der Sie den Zustand einfügen möchten. Alternative: Über das Kontextmenü des Diagramms: Klicken Sie mit der rechten Maustaste auf den Diagrammhintergrund. Wählen Sie im Kontextmenü **Hinzufügen > Zustand**.

Anmerkung: Sie können Zustände auch in bereits vorhandene Zustandselemente einfügen. Einzelne Zustände können auch ausgeblendet werden. So kann beispielsweise der Inhalt zusammengesetzter Zustände ausgeblendet werden, um das Diagramm übersichtlicher zu gestalten.

- Wenn Sie einen neuen Zustand in ein Diagramm einfügen, können Sie im Objektinspektor u.a. folgende Eigenschaften festlegen:

- Legen Sie die Standardeigenschaften des Elements fest.
- Wählen Sie im Feld **State Invariant** (Zustandsinvariante) die Sprache des Ausdrucks in der Liste **Sprache** aus. Die möglichen Optionen sind **OC** und **Text**.
- Überprüfen oder konfigurieren Sie auf der Registerkarte **Eigenschaften** das Verhalten des Zustands mit folgenden zusätzlichen Eigenschaften:

Feld	Beschreibung
Composite	Diese Eigenschaft ist True , wenn der Zustand eine oder mehrere Regionen enthält. Die Einstellung kann nicht geändert werden.
Orthogonal	Diese Eigenschaft ist True , wenn der Zustand zwei oder mehrere Regionen enthält. Die Einstellung kann nicht geändert werden.
Simple	Diese Eigenschaft ist True , wenn der Zustand keine Regionen enthält. Die Einstellung kann nicht geändert werden.
Do activity	Legen Sie im Objektinspektor die Aktivität fest, die bei der Ausführung des aktuellen Zustands durchgeführt wird. Sie können die gewünschte Aktivität in einem Aktivitätsdiagramm des Projekts auswählen.
Entry	Legen Sie im Objektinspektor die Aktivität fest, die beim Start der Ausführung des aktuellen Zustands durchgeführt wird. Sie können die gewünschte Aktivität in einem Aktivitätsdiagramm des Projekts auswählen.
Exit	Legen Sie im Objektinspektor die Aktivität fest, die am Ende der Ausführung des aktuellen Zustands durchgeführt wird. Sie können die gewünschte Aktivität in einem Aktivitätsdiagramm des Projekts auswählen.

Geben Sie den OCL-Ausdruck für den Zustand in das Textfeld unter der Liste ein.

Siehe auch

Überblick zur OCL-Unterstützung (siehe Seite 1453)

UML 2.0-Zustandsmaschinendiagramm (siehe Seite 1321)

1.163 UML 2.0-Zustandsmaschinendiagramme erstellen

Folgende Tipps und Techniken helfen Ihnen bei der Arbeit mit UML 2.0-Zustandsmaschinendiagrammen.

Gehen Sie zur Erstellung eines UML 2.0-Zustandsmaschinendiagramms folgendermaßen vor:

1. Erstellen Sie den Anfangs- und Endknoten.
2. Erstellen Sie Haupt- und Unterzustände.
3. Erstellen Sie Regionen.
4. Erstellen Sie Eintritts- und Austrittspunkte.
5. Erstellen Sie Pins.
6. Erstellen Sie Übergänge.
7. Erstellen Sie Historienknoten.
8. Wahlweise können Sie auch Verknüpfungen zu Elementen in anderen Diagrammen erstellen.

Siehe auch

Verknüpfungen erstellen (↗ siehe Seite 139)

UML 2.0-Zustandsmaschinendiagramme – Referenz (↗ siehe Seite 1321)

1.164 Navigation im Diagramm mit der Übersicht

So öffnen Sie den Übersichtsbereich:

1. Öffnen Sie ein Diagramm, und klicken Sie auf die Schaltfläche Übersicht. Der Bereich wird erweitert und zeigt eine Miniaturansicht des aktuellen Diagramms an.
2. Klicken Sie auf den schattierten Bereich, und ziehen Sie ihn. Auf diese Weise können Sie bequem einen Bildlauf im Diagramm durchführen.
3. Sie können die Größe der Übersicht ändern, indem Sie auf die obere linke Ecke klicken und ziehen, bis sie die gewünschte Größe hat.
4. Um den Übersichtsbereich zu schließen, klicken Sie auf das Diagramm.

Siehe auch

Diagramm zoomen ( siehe Seite 195)

1.165 Modellelemente aus- und einblenden

Sie können die Sichtbarkeit von Elementen in einem Diagramm mit dem Befehl Ausblenden im Kontextmenü einzelner Diagrammelemente sowie mit dem Befehl Ein-/Ausblenden im Kontextmenü des Diagramms steuern.

Sie können Elemente auf folgende Arten ausblenden:

1. Öffnen Sie die Diagrammansicht.

2. Verwenden Sie eines der folgenden Verfahren:

- Klicken Sie im Diagramm mit der rechten Maustaste auf das Element, und wählen Sie im Kontextmenü den Befehl Ausblenden.
- Wählen Sie im Diagramm mehrere Elemente aus (durch Klicken mit gedrückter STRG-Taste oder Aufziehen eines Auswahlrahmens), klicken Sie mit der rechten Maustaste auf die Auswahl, und wählen Sie im Kontextmenü den Befehl Ausblenden.
- Klicken Sie mit der rechten Maustaste auf den Diagrammhintergrund, und wählen Sie im Kontextmenü den Befehl Ein-/Ausblenden. Das Dialogfeld **Ausgeblendete anzeigen** wird geöffnet (siehe unten).

So blenden Sie Diagrammelemente mit dem Dialogfeld Ausgeblendete anzeigen ein oder aus:

1. Klicken Sie mit der rechten Maustaste auf das Diagramm, und wählen Sie im Kontextmenü den Befehl Ein-/Ausblenden. Das Dialogfeld Ausgeblendete anzeigen wird geöffnet.

2. Wählen Sie das bzw. die auszublenden Elemente in der Liste **Diagrammelemente** aus.

3. Sie haben folgende Möglichkeiten, Elemente von der Liste **Diagrammelemente** in die Liste **Ausgeblendete Elemente** zu übertragen:

- Doppelklicken Sie auf das Element.
- Klicken Sie einmal auf das Element, und klicken Sie dann auf **Hinzufügen**.
- Klicken Sie bei gedrückter STRG-Taste auf mehrere Elemente und danach auf die Schaltfläche **Hinzufügen**.

4. Auf folgende Arten können Sie Diagrammelemente aus der Liste **Ausgeblendete Elemente** entfernen:

- Doppelklicken Sie auf das Element.
- Klicken Sie auf das Element und danach auf die Schaltfläche Entfernen.
- Klicken Sie bei gedrückter STRG-Taste auf mehrere Elemente und danach auf die Schaltfläche Entfernen.
- Um alle Elemente aus der Liste **Ausgeblendete Elemente** zu entfernen, klicken Sie auf Alle entfernen.

5. Klicken Sie auf OK, um das Dialogfeld zu schließen.

Siehe auch

Ansichtsfilter verwenden (siehe Seite 194)

Ein Element erstellen (siehe Seite 142)

1.166 Ansichtsfilter verwenden

Die globalen Einstellungen für die Diagrammansicht werden über die Filter im Dialogfeld Optionen vorgenommen.

So aktivieren und deaktivieren Sie Ansichtsfilter:

1. Wählen Sie im Hauptmenü **Tools**▶**Optionen**.
2. Klicken Sie auf den Ordner Together.
3. Klicken Sie unter dem Knoten **(Ebene)**▶**Diagramm** auf **Ansichtsfilter**.

Anmerkung: Die Filter im Dialogfeld **Optionen** sind global. Wenn Sie speziell die Klassen filtern möchten, setzen Sie die Eigenschaft *Member anzeigen* auf **False**.

So filtern Sie Klassen:

1. Klicken Sie auf der Seite **Ansichtsfilter** des Dialogfelds Optionen auf das Feld **Member anzeigen**.
2. Klicken Sie auf den Pfeil der Dropdown-Liste, und wählen Sie **False**.
3. Klicken Sie auf OK.

Die Member und inneren Klassifizierer (Klassen, Delegaten, Aufzählungen, Interfaces und Strukturen) werden nun nicht mehr angezeigt.

Da die inneren Klassifizierer als Member des Container-Elements behandelt werden, wirken sich folgende Filter nicht auf sie aus:

Ansichtsfilter
Klassen anzeigen
Delegaten anzeigen
Aufzählungen anzeigen
Interfaces anzeigen
Strukturen anzeigen

Anmerkung: Quelltextspezifische Elemente stehen nur in Implementierungsprojekten zur Verfügung.

Siehe auch

Modellelemente aus- und einblenden (↗ siehe Seite 193)

Together-Diagrammoptionen (Ansichtsfilter) (↗ siehe Seite 1289)

1.167 Diagramme zoomen

Mit Hilfe des Kontextmenüs des Diagramms können Sie in der Diagrammansicht die gewünschte Vergrößerung einstellen.

So legen Sie die Vergrößerung in der Diagrammansicht fest:

1. Klicken Sie mit der rechten Maustaste auf den Diagrammhintergrund.
2. Wählen Sie im Kontextmenü den Befehl Zoom.
3. Wählen Sie den gewünschten Befehl im Untermenü.

Siehe auch

Zoom-Tastaturkürzel ( siehe Seite 1278)

1.168 OCL-Einschränkungen erstellen

So erstellen Sie eine Objekteinschränkung und stellen eine Beziehung mit dem Kontext her:

1. Klicken Sie in der Tool-Palette eines Klassen-/Paketdiagramms auf die Schaltfläche **Einschränkung** und danach auf den Diagrammhintergrund. Das neue Element wird mit aktiviertem OCL-Editor eingefügt.

2. Geben Sie den Einschränkungsausdruck ein.

3. Schließen Sie den OCL-Editor.

4. Klicken Sie in der Tool-Palette des Diagramms auf die Schaltfläche **Verbinden**, und verbinden Sie den Einschränkungsknoten mit dem gewünschten Designelement.

Tip: Das eingeschränkte Attribut muss im Kontext bereits vorhanden sein. Andernfalls wird die Einschränkung als ungültig gekennzeichnet.

Führen Sie alternativ folgende Schritte aus:

1. Klicken Sie in der Modellansicht oder im Diagramm mit der rechten Maustaste auf das Element, für das die Einschränkung erstellt werden soll.

2. Klicken Sie im Kontextmenü auf **Einschränkungen**.

3. Klicken Sie im Dialogfeld **Einschränkungen hinzufügen/entfernen** auf die Schaltfläche **Hinzufügen**.

4. Geben Sie die Einschränkung ein:

- Geben Sie den Namen der Einschränkung in das Feld **Name** ein.
- Wählen Sie im Feld **Sprache** den gewünschten Wert aus (**OC**L oder **T**ext).
- Geben Sie den Einschränkungstext in das Feld **Einschränkung** ein.

5. Fügen Sie ggf. weitere Einschränkungen hinzu.

6. Klicken Sie anschließend auf OK.

Siehe auch

Überblick zur OCL-Unterstützung (siehe Seite 1453)

Kombinierte Fragmente verwenden (siehe Seite 183)

OCL-Editor (Diagrammansicht) (siehe Seite 1269)

1.169 OCL-Ausdrücke bearbeiten

So aktivieren Sie den OCL-Editor:

1. Doppelklicken Sie auf ein Einschränkungselement oder eine OCL-Eigenschaft, oder wählen Sie ein Einschränkungselement aus, und drücken Sie die Taste F2. Der OCL-Editor wird geöffnet.
2. Bearbeiten Sie einen Ausdruck.
3. Klicken Sie auf die grüne Schaltfläche, um die Änderungen zu übernehmen und den OCL-Editor zu schließen. Wenn Sie auf die rote Schaltfläche klicken, wird der OCL-Editor geschlossen, ohne die Änderungen zu speichern.

Siehe auch

Überblick zur OCL-Unterstützung (siehe Seite 1453)

Kombinierte Fragmente verwenden (siehe Seite 183)

OCL-Einschränkungen erstellen (siehe Seite 196)

OCL-Editor (Diagrammansicht) (siehe Seite 1269)

1.170 OCL-Einschränkungen ein- und ausblenden

So blenden Sie eine einzelne Einschränkung aus:

1. Klicken Sie im Diagramm mit der rechten Maustaste auf eine Einschränkung.
2. Klicken Sie auf **Ausblenden**.

So blenden Sie mehrere Einschränkungen aus:

1. Klicken Sie mit der rechten Maustaste auf den Diagrammhintergrund.
2. Klicken Sie auf **Ein-/Ausblenden**.
3. Wählen Sie im Dialogfeld **Ausgeblendete anzeigen** die gewünschten Einschränkungen in der Liste **Diagrammelemente** aus.
4. Klicken Sie auf **Hinzufügen**.

So zeigen Sie ausgeblendete Einschränkungen an:

1. Klicken Sie mit der rechten Maustaste auf den Diagrammhintergrund.
2. Klicken Sie auf **Ein-/Ausblenden**.
3. Wählen Sie im Dialogfeld **Ausgeblendete anzeigen** die gewünschten Einschränkungen in der Liste **Verborgen** aus.
4. Klicken Sie auf **Entfernen**.

Siehe auch

Überblick zur OCL-Unterstützung (↗ siehe Seite 1453)

Kombinierte Fragmente verwenden (↗ siehe Seite 183)

OCL-Einschränkungen erstellen (↗ siehe Seite 196)

OCL-Ausdrücke bearbeiten (↗ siehe Seite 197)

OCL-Editor (Diagrammansicht) (↗ siehe Seite 1269)

1.171 Mit komplexen Zuständen arbeiten

Die folgenden Tipps helfen Ihnen bei der Arbeit mit komplexen zusammengesetzten Zuständen und Unterzuständen.

Die Größe des Hauptzustands kann geändert werden. Sie können auch einen Unterzustand erstellen, indem Sie ein Zustandsdiagramm in ein anderes Zustandsdiagramm einfügen und die Anfangs-, End- und Historienzustände sowie Übergänge angeben.

Sie können zusammengesetzte Zustände erstellen, indem Sie in einem Zustand eine oder mehrere Zustandsebenen verschachteln. Sie können in einen Zustand auch Anfangs-/Endzustände oder einen Historienzustand einfügen und Übergänge zwischen den Unterzuständen erstellen.

Ein zusammengesetzter (verschachtelter) Zustand kann auf folgende Arten erstellt werden.

1. Einen verschachtelten Unterzustand per Drag&Drop erstellen
2. Einen verschachtelten Unterzustand über das Kontextmenü des Zustandselements erstellen

So erstellen Sie einen verschachtelten Unterzustand per Drag&Drop:

1. Fügen Sie ein Zustandselement in das Diagramm ein.
2. Ziehen Sie einen neuen Zustand zu einem vorhandenen Zustand.
3. Lassen Sie die Maustaste los, um den neuen Zustand abzulegen.

So erstellen Sie einen verschachtelten Unterzustand über das Kontextmenü des Zustandselements:

1. Klicken Sie mit der rechten Maustaste auf den Zustand (eine Region), der (die) als Container fungieren soll.
2. Wählen Sie im Kontextmenü **Hinzufügen > Zustand**.

Tip: In einem Zustand können mehrere Ebenen von Unterzuständen verschachtelt werden. Bei besonders komplexen Zuständen ist es jedoch besser, für jeden Unterzustand ein eigenes Diagramm zu erstellen und die Diagramme dann sequenziell durch Hyperlinks zu verknüpfen.

Mit Hilfe des Befehls **Verknüpfungen** im Kontextmenü des Diagramms können Sie vorhandene Elemente aus anderen Zustandsdiagrammen verwenden. Klicken Sie mit der rechten Maustaste auf das Diagramm, und wählen Sie **Hinzufügen > Verknüpfungen**. Navigieren Sie dann in dem Bereich, in dem sich die Baumstruktur der verfügbaren Projektinhalte für die Projektgruppe befindet, zu dem vorhandenen Diagramm, und wählen Sie die gewünschten Elemente, Zustände, Historien, Aufspaltungen und/oder Zusammenführungen aus.

Tip: Bei Verwendung des Kontextmenüs des Zustandselements können Sie auch alle anderen Unterelemente erstellen, die der Zustand enthalten kann.

Tip: In einem Zustand kann nur ein Historienelement erstellt werden.

Siehe auch

Überblick zu Hyperlinks (siehe Seite 1450)

Verknüpfungen hinzufügen (siehe Seite 139)

UML 1.5- Aktivitätsdiagramm (siehe Seite 1308)

UML 1.5-Zustandsdiagramm (siehe Seite 1305)

UML 2.0-Zustandsmaschinendiagramm (siehe Seite 1321)

1.172 Verzögertes Ereignis erstellen

Verzögerte Ereignisse können einem Zustandselement hinzugefügt werden.

So erstellen Sie ein verzögertes Ereignis:

1. Wählen Sie in der Diagrammansicht oder der Modellansicht das gewünschte Zustands- oder Aktivitätselement aus.
2. Klicken Sie mit der rechten Maustaste auf das Element, und wählen Sie im Kontextmenü **Hinzufügen**▶**Verzögertes Ereignis**.

Siehe auch

Verzögerte Ereignisse ( siehe Seite 1308)

UML 1.5- Aktivitätsdiagramm ( siehe Seite 1308)

UML 1.5-Zustandsdiagramm ( siehe Seite 1305)

1.173 Interne Übergänge erstellen

So erstellen Sie einen internen Übergang:

1. Wählen Sie in der Diagrammansicht oder der Modellansicht das gewünschte Zustands- oder Aktivitätselement aus.
2. Klicken Sie mit der rechten Maustaste auf das Element, und wählen Sie im Kontextmenü **Hinzufügen > Interner Übergang**.

Siehe auch

- Mehrachübergänge erstellen (siehe Seite 202)
- Übergänge (siehe Seite 1307)
- UML 1.5- Aktivitätsdiagramm (siehe Seite 1308)
- UML 1.5-Zustandsdiagramm (siehe Seite 1305)
- UML 2.0-Zustandsmaschinendiagramm (siehe Seite 1321)

1.174 Mehrfachübergänge erstellen

So erstellen Sie einen Mehrfachübergang (eine Aufspaltung oder eine Zusammenführung):

1. Überlegen Sie sich, welche Zustände beteiligt sein sollen. Fügen Sie gegebenenfalls zuerst alle Zustände in das Diagramm ein, und ordnen Sie sie wie gewünscht an.
2. Klicken Sie in der Tool-Palette des Diagramms auf die Schaltfläche **Aufspaltung** oder **Zusammenführung**.
3. Fügen Sie eine vertikale oder horizontale Aufspaltung bzw. Zusammenführung in das Diagramm ein.
4. Nehmen Sie die entsprechenden Größenänderungen vor.
5. Klicken Sie in der Tool-Palette des Diagramms auf die Schaltfläche **Übergang**.
6. Erstellen Sie Beziehungen vom Quellzustand bzw. den Quellzuständen zum Aufspaltungs-/Zusammenführungsknoten sowie vom Aufspaltungs-/Zusammenführungsknoten zum Zielzustand bzw. zu den Zielzuständen.

Siehe auch

- Interne Übergänge erstellen ( siehe Seite 201)
- Übergänge ( siehe Seite 1307)
- UML 1.5- Aktivitätsdiagramm ( siehe Seite 1308)
- UML 1.5-Zustandsdiagramm ( siehe Seite 1305)
- UML 2.0-Zustandsmaschinendiagramm ( siehe Seite 1321)

1.175 Selbstübergänge erstellen

So erstellen Sie einen Selbstübergang:

1. Zeichnen Sie einen Übergang vom Zustands- oder Aktivitätselement, und ziehen Sie die Beziehung vom Element weg.
2. Ziehen Sie die Beziehung zurück zum Element, und legen Sie sie ab.

Alternative:

1. Zeichnen Sie einen Übergang zwischen zwei Aktivitäten (oder Zuständen).
2. Ziehen Sie das entgegengesetzte Ende der Beziehungslinie zurück zur gewünschten Aktivität (oder zum Zustand).

Siehe auch

Eine einfache Beziehung erstellen ( siehe Seite 141)

UML 1.5- Aktivitätsdiagramm ( siehe Seite 1308)

UML 1.5-Zustandsdiagramm ( siehe Seite 1305)

Tool-Palette ( siehe Seite 1267)

1.176 Eintritts- und Austrittsaktionen festlegen

Sie können Eintritts- und Austrittsaktionen als Knoten oder stereotypisierte **interne Übergänge** erstellen.

So legen Sie Eintritts- und Austrittsaktionen mit Hilfe des internen Editors fest:

1. Erstellen Sie im gewünschten Zustand einen internen Übergang.
2. Doppelklicken Sie auf den Übergang, um den internen Editor zu aktivieren.
3. Benennen Sie den internen Übergang entsprechend der folgenden Syntax um:

Stereotyp/Aktionsname (Argument)

Ein Beispiel:

```
exit/setState(idle)
```

So legen Sie Eintritts- und Austrittsaktionen mit Hilfe interner Übergänge fest:

1. Erstellen Sie den internen Übergang.
2. Weisen Sie im Objektinspektor die Eigenschaften für den Ereignisnamen, die Ereignisargumente und den Aktionsausdruck des internen Übergangs zu.

Siehe auch

UML 1.5-Aktivitätsdiagramm (siehe Seite 1308)

1.177 Mit Instanzspezifikationen arbeiten

Sie können einen Klassifizierer über den Objektinspektor oder den internen Editor instantiiieren.

Bei der Verwendung einer Instanzspezifikation haben Sie folgende Möglichkeiten:

1. Instantiiieren eines Klassifizierers im Objektinspektor.
2. Instantiiieren eines Klassifizierers mit dem internen Editor.
3. Definieren der Funktionsweise einer Instanzspezifikation.
4. Hinzufügen eines Slots zu einem Instanzspezifikationselement.
5. Zuordnen eines strukturellen Features zu einem Slot.
6. Zuweisen des Slot-Wertes.
7. Definieren des das Slot-Stereotyps.

So instantiiieren Sie einen Klassifizierer im Objektinspektor:

1. Wählen Sie im Diagramm eine Instanzspezifikation aus.
2. Wählen Sie im Knoten **Allgemein** des Objektinspektors das Feld **instantiates** aus.
3. Klicken Sie auf die Auswahlorschaltfläche.
4. Wählen Sie im Dialogfeld Klasse oder Interface für Typ auswählen die gewünschten Klassifizierer mit Hilfe der Schaltflächen Hinzufügen und Entfernen aus.
5. Klicken Sie auf OK, um den Vorgang abzuschließen.

So instantiiieren Sie einen Klassifizierer mit dem internen Editor:

1. Wählen Sie im Diagramm eine Instanzspezifikation aus.
2. Drücken Sie F2, um den internen Editor zu öffnen. Sie können auch auf den Namen der Instanzspezifikation doppelklicken.
3. Geben Sie nach dem Namen der Instanzspezifikation einen Doppelpunkt und danach den Namen eines vorhandenen Klassifizierers ein. Beispiel: InstanceSpecification1:Class1.
4. Drücken Sie die EINGABETASTE.

So definieren Sie die Funktionsweise einer Instanzspezifikation:

1. Fügen Sie Slots in ein Instanzspezifikationselement ein.
2. Verknüpfen Sie die Slots mit den Attributen der instantiierten Klassifizierer.
3. Legen Sie den Wert fest, und definieren Sie das Slot-Stereotyp.

So fügen Sie einem Instanzspezifikationselement einen Slot hinzu:

1. Fügen Sie ein Instanzspezifikationselement in das Diagramm ein.
2. Klicken Sie mit der rechten Maustaste auf das Instanzspezifikationselement, und wählen Sie im Kontextmenü **Neu > Slot**.

So ordnen Sie einem Slot ein strukturelles Feature zu:

1. Wählen Sie einen Slot in einem Instanzspezifikationselement aus.
2. Erweitern Sie den Knoten Allgemein im Objektinspektor.
3. Klicken Sie im Feld für die Definition des Features auf die Auswahlorschaltfläche.

4. Wählen Sie im Dialogfeld Attribut zum Definieren des Features auswählen das gewünschte Attribut aus, und klicken Sie auf OK.

So legen Sie den Slot-Wert fest:

1. Wählen Sie einen Slot aus.
2. Verwenden Sie eines der folgenden Verfahren:
 - Geben Sie im Objektinspektor des Slots den gewünschten Wert in das entsprechende Feld ein.
 - Öffnen Sie den internen Editor für den Slot, und geben Sie nach dem Slot-Namen zuerst ein Gleichheitszeichen und dann den gewünschten Wert ein.

So definieren Sie das Slot-Stereotyp:

1. Erweitern Sie im Objektinspektor des Slots den Knoten Allgemein.
2. Geben Sie den Wert in das Feld Stereotyp ein.

Siehe auch

- UML 2.0-Klassendiagramm ( siehe Seite 1313)
- UML 2.0-Interaktionsdiagramm ( siehe Seite 1315)
- UML 2.0-Komponentendiagramm ( siehe Seite 1326)
- UML 2.0-Kompositionssstrukturdiagramm ( siehe Seite 1329)

1.178 Mit bereitgestellten und erforderlichen Interfaces arbeiten

So erstellen Sie ein bereitgestelltes Interface:

1. Erstellen Sie mit den entsprechenden Schaltflächen der Tool-Palette ein Klassen- und ein Interface-Knotenelement.
2. Klicken Sie in der Tool-Palette des Diagramms auf die Schaltfläche **Interface**.
3. Klicken Sie auf die Client-Klasse, und ziehen Sie mit gedrückter Maustaste zum Interface-Knoten.

So erstellen Sie ein erforderliches Interface:

1. Erstellen Sie mit den entsprechenden Schaltflächen der Tool-Palette ein Klassen- und ein Interface-Knotenelement.
2. Klicken Sie in der Tool-Palette des Diagramms auf die Schaltfläche **Erforderliches Interface**.
3. Klicken Sie auf die Client-Klasse, und ziehen Sie mit gedrückter Maustaste zum Interface-Knoten.

Siehe auch

Das Erscheinungsbild von Interfaces ändern (siehe Seite 214)

UML 2.0-Klassendiagramm (siehe Seite 1313)

UML 2.0-Komponentendiagramm (siehe Seite 1326)

UML 2.0-Kompositionsstrukturdiagramm (siehe Seite 1329)

1.179 Assoziationsklasse erstellen

So erstellen Sie eine Assoziationsklasse:

1. Klicken Sie in der Tool-Palette des Diagramms auf die Schaltfläche **Assoziationsklasse**.
 2. Klicken Sie auf den Diagrammhintergrund. Dadurch wird ein mit dem Rautensymbol verbundenes Klassensymbol für die Assoziationsklasse eingefügt.
 3. Erstellen Sie die Teilnehmerklassen.
 4. Verbinden Sie mit Hilfe der Schaltfläche **Assoziationsende** die n-fache Assoziation mit den Teilnehmerklassen.
- Ergebnis: Der Quelltext einer Assoziationsklasse enthält entsprechende Tags für die Assoziationsklasse selbst und für alle Assoziationsendklassen.

So löschen Sie eine Assoziationsklasse:

1. Klicken Sie mit der rechten Maustaste auf die Assoziationsendebeziehung, die Assoziationsklasse, oder die Verbindung.
 2. Klicken Sie im Kontextmenü auf Gesamte Assoziationsklasse löschen.
- Ergebnis: Die gesamte Assoziationsklasse wird aus dem Diagramm entfernt.

Siehe auch

Klassendiagrammbeziehungen (siehe Seite 1296)

UML 2.0-Klassendiagramme (siehe Seite 1313)

UML 1.5-Klassendiagramme (siehe Seite 1294)

1.180 Innere Klassifizierer erstellen

Dieser Abschnitt enthält Anleitungen für das Hinzufügen von inneren Klassifizierern zu Klassen (einschließlich Windows-Klassen, wie Windows Forms, geerbte Formulare, User Controls usw.), Strukturen und Modulen (Containern) in Implementierungsprojekten.

Sie haben die Möglichkeit, innere Klassifizierer in der Diagrammansicht oder der Modellansicht über das entsprechende Kontextmenü des Diagrammelements zu Klassendiagrammelementen (Containern) hinzuzufügen. Sie können einen Klassifizierer auch in der Tool-Palette auswählen und in der Diagrammansicht auf das Container-Element klicken, um ihm den inneren Klassifizierer hinzuzufügen.

Anmerkung: Strukturelemente stehen nur für Implementierungsprojekte zur Verfügung.

Tip: Ein innerer Klassifizierer kann per Drag&Drop oder mit einer Zwischenablage-Operation aus einem Container-Element entfernt werden.

So erstellen Sie einen inneren Klassifizierer mit Hilfe des Kontextmenüs:

1. Klicken Sie mit der rechten Maustaste auf das Container-Element.
2. Wählen Sie **Hinzufügen ▶ (Innerer_Klassifizierer_Typ)**. Die Definition von (Innerer_Klassifizier_Typ) finden Sie in der obigen Tabelle.

So erstellen Sie einen inneren Klassifizierer mit den Zwischenablage-Operationen Ausschneiden, Kopieren und Einfügen:

1. Kopieren oder schneiden Sie einen vorhandenen Klassifizierer mit der entsprechenden Zwischenablage-Operation aus.
2. Wählen Sie das gewünschte Container-Element aus.
3. Fügen Sie den Klassifizierer mit der Zwischenablage-Operation **Einfügen** in das Container-Element ein.

So erstellen Sie einen inneren Klassifizierer per Drag&Drop:

1. Wählen Sie einen Klassifizierer in der Diagrammansicht aus.
2. Ziehen Sie ihn per Drag&Drop in einen Container in der Diagrammansicht. Die Position, die Together als gültiges Ziel für das Ablegen eines inneren Klassifizierers akzeptiert, wird mit einem blauen Rahmen hervorgehoben.

Siehe auch

Ein Element erstellen (siehe Seite 142)

UML 1.5-Klassendiagramm (siehe Seite 1294)

UML 2.0-Klassendiagramm (siehe Seite 1313)

Innere Klassifizierer (siehe Seite 1298)

1.181 Klassendiagramme als Ansichten verwenden

Mit Hilfe von Klassendiagrammen lassen sich Unteransichten des Projekts erzeugen.

So verwenden Sie ein Klassendiagramm als Ansicht:

1. Erstellen Sie ein neues Klassendiagramm.
2. Erzeugen Sie Verknüpfungen zum Originaldiagramm, um schnell und einfach Unteransichten zu erstellen, die eine einfachere Verwaltung ermöglichen.

Tip: Diese Funktion bietet sich auch an, um Ansichten verteilter Klassen in einem Diagramm zusammenzuführen. Together zeigt dann automatisch alle Beziehungen an, die zwischen diesen Klassen bestehen.

Anmerkung: In Implementierungsprojekten werden die Änderungen, die Sie im Diagramm vornehmen, in den Quelltext übernommen. Diagramm und Quelltext bleiben dadurch synchron.

Siehe auch

LiveSource – Überblick ([siehe Seite 1451](#))

UML 1.5-Klassendiagramm ([siehe Seite 1294](#))

UML 2.0-Klassendiagramm ([siehe Seite 1313](#))

1.182 Mit Interfaces arbeiten

So erstellen Sie ein Interface:

1. Erstellen Sie mit den entsprechenden Schaltflächen der Tool-Palette ein Klassen- und ein Interface-Knotenelement.
2. Klicken Sie in der Tool-Palette des Diagramms auf die Schaltfläche **Generalisierungsbeziehung**.
3. Klicken Sie auf die Client-Klasse, und ziehen Sie mit gedrückter Maustaste zum Interface-Knoten.

So blenden Sie ein Interface aus:

1. Wählen Sie ein Interface aus.
2. Klicken Sie mit der rechten Maustaste, und wählen Sie im Kontextmenü den Befehl Objektinspektor.
3. Erweitern Sie den Knoten Ansicht.
4. Aktivieren Sie die Option **Ausblenden**.

Um alle Interfaces auszublenden, deaktivieren Sie den Ansichtsfilter Interfaces anzeigen.

Siehe auch

Das Erscheinungsbild von Interfaces ändern ( siehe Seite 214)

UML 1.5-Klassendiagramm ( siehe Seite 1294)

UML 2.0-Klassendiagramm ( siehe Seite 1313)

1.183 Einem Container ein Member hinzufügen

Sie können Member zu Klassendiagrammelementen (Containern) über das Kontextmenü des Diagrammelements in der Diagrammansicht bzw. der Modellansicht oder über die entsprechenden Tastaturbefehle hinzufügen.

So fügen Sie einem Container ein Member hinzu:

1. Klicken Sie mit der rechten Maustaste auf den Container (Klasse, Interface usw.).

2. Wählen Sie **Hinzufügen ▾ (Member_Typ)**. Die Definition von **(Member_Typ)** finden Sie in der obigen Tabelle.

Tip: Sie können auch mit Hilfe von Tastaturkürzeln Felder und Methoden in einen Container einfügen, für den die betreffenden Member zulässig sind. Drücken Sie **STRG+W** (für Felder) oder **STRG+M** (für Methoden bzw. Funktionen).

3. Sie können das Member mit Hilfe des internen Editors, des Objektinspektors oder des Quelltext-Editors bearbeiten.

Ergebnis: Das neue Member wird entsprechend der für Diagramme gültigen Element-Sortierreihenfolge in den Bereich des Containers eingefügt. Die Sortierreihenfolge kann im Dialogfeld Optionen festgelegt werden.

Tip: Wenn ein Container bereits Member enthält, können Sie mit der rechten Maustaste auf ein Member klicken und über das Kontextmenü ein weiteres Member erzeugen. Sie können das Member auch auswählen und die Taste **EINFG** drücken.

Siehe auch

Ein Element erstellen (siehe Seite 142)

Member (verfügbar zum Hinzufügen) (siehe Seite 1299)

UML 1.5-Klassendiagramme (siehe Seite 1294)

UML 2.0-Klassendiagramme (siehe Seite 1313)

1.184 Das Erscheinungsbild von Abschnitten ändern

Sie können Abschnitte für Member von Klassen, Interfaces, Namespaces, Aufzählungen und Strukturen (nur C#-Projekte) ein- und ausblenden. Standardmäßig werden die Abschnitte für diese Elemente im Diagramm als gerade Linie dargestellt. In Dialogfeld Optionen können Sie die Anzeigevoreinstellungen für Abschnittssteuerelemente festlegen. Das Hinzufügen von Abschnittssteuerelementen ist sinnvoll, wenn Sie mit großen Container-Elementen arbeiten, deren Inhalt nicht ständig sichtbar sein muss.

So blenden Sie Abschnitte ein und aus:

1. Wählen Sie die Klasse (oder das Interface) im Diagramm aus.
2. Klicken Sie auf das + oder - in der linken Ecke des Abschnitts.

So zeigen Sie die Abschnittssteuerelemente an:

1. Öffnen Sie das Dialogfeld **Optionen**.
2. Wählen Sie die Kategorie **Together** ▶ **(Ebene)** ▶ **Diagramm** ▶ **Erscheinungsbild** ▶ **Knoten**.
3. Ändern Sie in dieser Kategorie den Wert des Feldes Abschnitte als Linie anzeigen.

Siehe auch

- UML 2.0-Klassendiagramm ([siehe Seite 1313](#))
- UML 1.5-Klassendiagramm ([siehe Seite 1294](#))
- Diagrammoptionen (Erscheinungsbild) ([siehe Seite 1282](#))

1.185 Das Erscheinungsbild von Interfaces ändern

So zeigen Sie ein Interface mit Hilfe des Kontextmenüs als Kreis an:

1. Klicken Sie in der Diagrammansicht oder der Modellansicht mit der rechten Maustaste auf das Interface-Element.
2. Wählen Sie den Befehl Als Kreis anzeigen.

Tip: Dieser Befehl funktioniert wie ein Schalter. Klicken Sie erneut mit der rechten Maustaste auf das Element, und wählen Sie Als Kreis anzeigen, um das Interface-Element wieder als Rechteck darzustellen.

Anmerkung: Wenn Interfaces als kleine Kreise dargestellt werden, sind ihre Member in der Diagrammansicht nicht zu sehen. Verwenden Sie zur Anzeige der Member die Modellansicht.

So zeigen Sie ein Interface mit Hilfe des `color="ide"`Objektinspektors als Kreis an:

1. Wählen Sie das Interface-Element in der Diagrammansicht oder der Modellansicht aus.
2. Drücken Sie die Taste F4, um den Objektinspektor zu öffnen.
3. Setzen Sie die Eigenschaft Circle view (Kreisansicht) auf **True**.

Tip: Wenn das Interface-Element wieder als Rechteck dargestellt werden soll, setzen Sie die Eigenschaft auf **False**.

Siehe auch

Notation ändern (siehe Seite 117)

1.186 Mit Konstruktoren arbeiten

Sie können beliebig viele Konstruktoren in einer Klasse erstellen.

In Designprojekten wird ein Konstruktor als Operation mit dem Stereotyp <<constructor>> erstellt.

In Implementierungsprojekten wird jeder neue Konstruktor mit einem eindeutigen Satz von Parametern erzeugt. Sie können außerdem mit Hilfe des Objektinspektors benutzerdefinierte Parameter festlegen.

Tip: Konstruktoren und Destruktoren lassen sich wie alle anderen Member zwischen Container-Klassen verschieben, kopieren und einfügen.

So definieren Sie die Konstruktorparameter (nur bei Implementierungsprojekten):

1. Wählen Sie den gewünschten Konstruktor in einer Klasse aus.
2. Klicken Sie im Objektinspektor auf das Feld **Params**.
3. Geben Sie in das Textfeld die Liste der Parameter mit dem bisherigen Typnamen ein. Verwenden Sie ein Komma als Begrenzer.

Siehe auch

UML 2.0-Klassendiagramm (siehe Seite 1313)

UML 1.5-Klassendiagramm (siehe Seite 1294)

1.187 Mit Feldern arbeiten

Dieses Thema betrifft nur Implementierungsprojekte.

Im Quelltext können mehrere Felder in einer Zeile deklariert werden. Im Diagramm werden die Felder dann in Form mehrerer Einträge im Abschnitt **Felder** eines Klassensymbols dargestellt. Wenn Sie ein Feld umbenennen, Modifizierer ändern, Anfangswerte festlegen usw., werden diese Änderungen auf das entsprechende Feld im Diagrammsymbol angewendet. Sie können diese Felder über Kontextmenübefehle oder Drag&Drop-Operationen im Diagramm kopieren und verschieben. Eingefügte Felder werden dann im Ziel-Container separat angezeigt.

So benennen Sie ein Feld um:

1. Wählen Sie ein Feld aus.
2. Geben Sie den neuen Namen im internen Editor der Diagrammansicht oder der Modellansicht, in das Feld **Name** des Objektinspektors oder im Quelltext-Editor ein.

So definieren Sie den Sichtbarkeitsmodifizierer:

1. Wählen Sie ein Feld aus.
2. Geben Sie das Sichtbarkeitsattribut im internen Editor der Diagrammansicht ein, wählen Sie es im Kombinationsfeld **Visibility** (Sichtbarkeit) des Objektinspektors aus, oder nehmen Sie die Änderung im Quelltext-Editor vor.

So definieren Sie das Stereotyp eines Feldes:

1. Wählen Sie ein Feld aus.
2. Verwenden Sie den internen Editor in der Diagrammansicht, das Kombinationsfeld **Stereotype** im Objektinspektor oder den Quelltext-Editor.

So definieren Sie Modifizierer, Anfangswerte, zugeordnete Objekte usw.:

1. Wählen Sie ein Feld aus.
2. Verwenden Sie den Objektinspektor oder den Quelltext-Editor.

Auf diese Weise bleiben Modell und Quelltext synchron.

Siehe auch

Modellansicht (siehe Seite 262)

Ein Element erstellen (siehe Seite 142)

UML 1.5-Klassendiagramme (siehe Seite 1294)

UML 2.0-Klassendiagramme (siehe Seite 1313)

1.188 Mit Beziehungen arbeiten

Sie können den Typ einer Assoziationsbeziehung ändern.

So erstellen Sie eine Assoziationsbeziehung:

1. Verwenden Sie die Schaltfläche **Assoziationsbeziehung** in der Tool-Palette des UML-Klassendiagramms, um Assoziationsbeziehungen zwischen Diagrammelementen zu definieren.
2. Der Beziehungstyp (Assoziation, Aggregation oder Komposition) und die Kardinalität von Client und Anbieter (Supplier) können im Objektinspektor festgelegt werden.
3. Sie können den Beziehungstyp auch über das Kontextmenü der Beziehung angeben. Wenn Sie eine Assoziationsbeziehung erzeugen, bestimmt Together ein Feld in der Client-Klasse (Ausgangspunkt der Beziehung).

So definieren Sie eine gerichtete Assoziationsbeziehung:

1. Wählen Sie **Ansicht** | Objektinspektor, wenn der Objektinspektor noch nicht geöffnet ist.
2. Wählen Sie eine Beziehung im Diagramm aus. Die Eigenschaften für die Beziehung werden im Objektinspektor angezeigt.
3. Klicken Sie im Objektinspektor auf das Feld **Directed**.
4. Klicken Sie auf den Pfeil der Dropdown-Liste, und wählen Sie *Directed* (**True** oder **False**) aus der Liste.

Siehe auch

Eine einfache Beziehung erstellen (siehe Seite 141)

Den Typ einer Beziehung ändern (siehe Seite 121)

UML 1.5-Klassendiagramme (siehe Seite 1294)

UML 2.0-Klassendiagramme (siehe Seite 1313)

1.189 Nachrichtenbeziehungen einer Methode zuordnen

Nachrichtenbeziehungen können den Methoden der Empfängerklasse zugeordnet werden. Die betreffenden Methoden werden entweder aus einer Liste gewählt oder erstellt. Zu diesem Zweck stehen zwei Befehle im Kontextmenü der Nachricht zur Verfügung: Hinzufügen und Auswählen.

Mit Hilfe des Feldes Operation im Objektinspektor kann die Methode umbenannt werden. In einem Dialogfeld werden Sie gefragt, ob Sie eine neue Methode erstellen oder die vorhandene umbenennen möchten.

Sie können die im Folgenden beschriebenen Verfahren verwenden, um einer Nachrichtenbeziehung eine Methode (Operation) zuzuordnen.

1. Erstellen einer neuen Methode für eine vorhandene Nachrichtenbeziehung.
2. Zuweisen einer Nachrichtenbeziehung zu einer vorhandenen Methode.
3. Aufheben der Methodenzuordnung.

So erstellen Sie eine neue Methode für eine vorhandene Nachrichtenbeziehung:

1. Definieren Sie eine Nachrichtenbeziehung zwischen zwei Objekten. Das Empfängerobjekt muss eine Klasse instantiiieren.
2. Wählen Sie im Kontextmenü der Nachrichtenbeziehung den Befehl Hinzufügen. Im angezeigten Untermenü haben Sie die Wahl zwischen Methode, Konstruktor und Destruktor.

Anmerkung: Destruktoren stehen nur für Klassen in C#-Projekten zur Verfügung.

3. Wählen Sie den gewünschten Operationstyp im Untermenü aus.

Tip: Wenn das Empfängerobjekt keine Klasse instantiiert, ist der Befehl **Hinzufügen** im Kontextmenü nicht verfügbar. Ist das Empfängerobjekt mit einem Interface verknüpft, können der Nachrichtenbeziehung nur Methoden zugeordnet werden.

Ergebnis: Die neue Operation wird in der Klasse des Empfängerobjekts erstellt. Der Nachrichtenbeziehung wird entsprechend des Operationstyps ein Operationsname zugewiesen:

Wenn eine Methode ausgewählt ist, lautet die Bezeichnung `Method<n> () : return_type`.

Ist ein Konstruktor ausgewählt, lautet die Bezeichnung `<Classname>()` in C#-Projekten.

Bei Auswahl eines Destruktors lautet die Bezeichnung `~<Classname>()`. Die Option **Destruktor** ist im Untermenü des Befehls Hinzufügen deaktiviert.

Mit Hilfe des Feldes Operation im Objektinspektor kann eine neue Methode im Klassifizierer erstellt werden. Sie können beispielsweise `Methodenname(Parametertypen) : Rückgabetyp` in das Feld Operation eingeben. Die Eingabe von Parametertypen ist optional. Wenn die Methode in der Klasse nicht vorhanden ist, werden Sie in einem Dialogfeld zur Erstellung der Methode aufgefordert. Ist die Methode bereits vorhanden, wird die Nachrichtenbeziehung automatisch für diese Methode eingerichtet.

So weisen Sie einer vorhandenen Methode eine Nachrichtenbeziehung zu:

1. Definieren Sie eine Nachrichtenbeziehung zwischen zwei Objekten. Das Empfängerobjekt muss eine Klasse instantiiieren.
2. Wählen Sie im Kontextmenü der Nachrichtenbeziehung den Befehl Methode auswählen. Im Untermenü wird die Liste mit den

Operationen der Empfängerklasse angezeigt.

3. Wenn die erforderliche Operation nicht in der Liste enthalten ist, klicken Sie auf Weitere, um die nächsten 20 Methoden (einschließlich geerbter Operationen) der Empfängerklasse anzuzeigen.

4. Wählen Sie die gewünschte Operation aus.

Ergebnis: Die zugeordnete Operation wird in der Liste der Methoden, Konstruktoren und Destruktoren ausgewählt.

Wenn Sie einen Klassifizierer mit einem Objekt verknüpfen, das bereits mit einem Klassifizierer instantiiert ist, werden alle Nachrichtenbeziehungen, für die die Eigenschaft Operation gesetzt ist, automatisch als Text gespeichert. Dies gilt nicht, wenn die Signatur der Methode mit der Signatur einer anderen Methode im neu verknüpften Klassifizierer identisch ist.

So heben Sie die Methodenzuordnung auf:

1. Wählen Sie die Nachrichtenbeziehung aus.

2. Wählen Sie im Kontextmenü der Nachrichtenbeziehung den Befehl Methodenzuordnung aufheben.

Ergebnis: Die Assoziation zwischen der Nachrichtenbeziehung und der Operation wird entfernt. Die Operation bleibt jedoch in der Empfängerklasse erhalten.

Wenn Sie die Zuordnung eines Klassifizierers zu einem Objekt aufheben und das Objekt über Nachrichtenbeziehungen verfügt, deren Eigenschaft Operation eine Methode des nicht mehr verknüpften Klassifizierers zugewiesen ist, wird ein Dialogfeld geöffnet. Darin werden Sie aufgefordert, entweder die Zuordnung zwischen Methode und Nachrichtenbeziehung aufzuheben oder die Nachrichtenbeziehung als Text zu speichern. Wenn Sie sich für das Speichern als Text entscheiden, wird die Eigenschaft Operation in Anführungszeichen gesetzt und die Operation im Diagramm rot hervorgehoben. Diese Funktion dient dazu, die Signaturen aller Methoden zu erhalten, die mit den Nachrichtenbeziehungen verknüpft sind. Wenn Sie das Objekt erneut mit einer Klasse instantiiieren, können Sie die Anführungszeichen löschen. In diesem Fall wird ein Dialogfeld geöffnet, in dem Sie zur Erstellung der Methode aufgefordert werden, sofern sie nicht im verknüpften Klassifizierer vorhanden ist. Das Dialogfeld wird nicht angezeigt, wenn die Signatur der Methode mit der einer im Klassifizierer vorhandenen Methode übereinstimmt.

Siehe auch

Eine einfache Beziehung erstellen ([siehe Seite 141](#))

UML 1.5-Interaktionsdiagramme ([siehe Seite 1302](#))

1.190 Eine Diagrammabfolge erstellen

Mit Hilfe der Hyperlink-Funktion von Together können Sie Abfolgen von beliebigen Anwendungsfalldiagrammen (und anderen Diagrammen) erstellen und diese dann nacheinander anzeigen.

So erstellen Sie eine Diagrammabfolge:

1. Sie können ganze Diagramme auf einer Detailebene mit dem nächsten Diagramm auf einer niedrigeren bzw. höheren Ebene verknüpfen oder eine Verknüpfung zwischen den Hauptanwendungsfällen oder -akteuren und dem nächsten Diagramm erstellen. Wenn Sie dann die Hyperlink-Sequenz durchlaufen, können Sie die Beziehungen zwischen den Diagrammen nachverfolgen.
2. Die Hyperlink-Funktion von Together ist aber nicht auf solche Sequenzen beschränkt. Sie können Diagramme und Elemente entsprechend Ihren jeweiligen Bedürfnissen per Hyperlinks miteinander verknüpfen. Wenn Sie beispielsweise eine hierarchische Abfolge der Anwendungsfalldiagramme erstellen und dann die entsprechenden Hyperlinks erstellen, können Sie einem bestimmten Akteur durch alle Anwendungsfälle folgen, in denen er verwendet wird.

Siehe auch

Überblick zu Hyperlinks ([siehe Seite 1450](#))

UML 1.5-Anwendungsfalldiagramm ([siehe Seite 1300](#))

UML 2.0-Anwendungsfalldiagramm ([siehe Seite 1314](#))

1.191 Erweiterungspunkte erstellen

So erstellen Sie einen Erweiterungspunkt:

1. Klicken Sie mit der rechten Maustaste auf das Anwendungsfallelement.
2. Klicken Sie im Kontextmenü auf **Hinzufügen > Erweiterungspunkt**.
3. Geben Sie einen Namen ein.

Siehe auch

UML 1.5-Anwendungsfalldiagramm (↗ siehe Seite 1300)

UML 2.0-Anwendungsfalldiagramm (↗ siehe Seite 1314)

1.192 Eine Anwendungsfalldiagramm erstellen

Ein Anwendungsfalldiagramm beschreibt in der Regel ein System und dessen Anforderungen.

So erstellen Sie eine Anwendungsfalldiagramm:

1. Normalerweise werden zuerst die Hauptanwendungsfälle des Systems definiert.
2. Danach folgt eine feinere Abstufung. So könnte beispielsweise der Hauptanwendungsfall "Auftragsverwaltung" eine weitere Detailebene mit Anwendungsfällen wie "Kunden eingeben" und "Verkäufe eingeben" enthalten.
3. Nachdem Sie diese feinere Abstufung erreicht haben, ist es hilfreich, eine Übersicht über den Gültigkeitsbereich und die Beziehungen der verschiedenen Anwendungsfalldiagramme des Systems zu erhalten.

Siehe auch

UML 1.5-Anwendungsfalldiagramm ([siehe Seite 1300](#))

UML 2.0-Anwendungsfalldiagramm ([siehe Seite 1314](#))

1.193 Funktion zur Dokumentationserzeugung konfigurieren

Im Dialogfeld Optionen können Sie Titel, Kopfzeile, Fußzeile und andere spezifische Einstellungen für die Dokumentation festlegen.

Eine Beschreibung der verfügbaren Optionen finden Sie im Dialogfeld Optionen und in der vorliegenden Online-Hilfe.

So konfigurieren Sie die Funktion für die Dokumentationserzeugung:

1. Wählen Sie im Hauptmenü **Tools**▸**Optionen**▸**Together**▸**(Ebene)**▸**Dokumentation erzeugen**.
2. Erweitern Sie die Kategorie **Allgemein**, und geben Sie Werte für den Titel der Dokumentation, den Fenstertitel, die Kopfzeile und die Fußzeile ein.
3. Legen Sie mit der Option **Internen Browser verwenden** fest, ob die erzeugte Dokumentation in einem externen Browser oder im internen Browser von RAD Studio geöffnet wird. Per Vorgabe wird die Dokumentation im externen Browser geöffnet.
4. Wählen Sie unter der Kategorie **Einbeziehen** die Sichtbarkeitsmodifizierer für Klassen und Member aus, die in die Dokumentation aufgenommen werden sollen.
5. Legen Sie unter **Navigation** die Optionen für die Erzeugung von Navigationsleiste, Index, Klassenhierarchie und Hilfe-Link fest.

Siehe auch

Überblick zur Dokumentationserzeugung (☞ siehe Seite 1458)

Projektdokumentation erzeugen (☞ siehe Seite 224)

Together-Optionen für die Dokumentationserzeugung – Referenz (☞ siehe Seite 1291)

1.194 Projektdokumentation erzeugen

So generieren Sie Projektdokumentation:

1. Wählen Sie in der Modellsicht ein Projekt, einen Namespace oder ein Diagramm aus.
2. Wählen Sie im Hauptmenü **Tools > Dokumentation erzeugen**. Sie können auch mit der rechten Maustaste auf die Auswahl klicken und im Kontextmenü den Befehl Dokumentation erzeugen wählen.
3. Legen Sie im Dialogfeld Dokumentation erzeugen die gewünschten Bereichs- und Optionseinstellungen fest.
4. Klicken Sie auf OK, um die Dokumentation zu erzeugen. In der Standardeinstellung erzeugt der Experte die Dokumentation für das gesamte Projekt.

Siehe auch

Überblick zur Dokumentationserzeugung (↗ siehe Seite 1458)

Funktion zur Dokumentationserzeugung konfigurieren (↗ siehe Seite 223)

Together-Optionen für die Dokumentationserzeugung – Referenz (↗ siehe Seite 1291)

1.195 Mit Namespaces und Paketen arbeiten

Namespaces werden in Implementierungsprojekten, Pakete in Designprojekten verwendet.

Folgende Aktionen können mit Namespaces und Paketen ausgeführt werden:

1. Namespace oder Paket anzeigen
2. Namespace oder Paket öffnen
3. Namespace oder Paket löschen
4. Namespace oder Paket umbenennen

So zeigen Sie einen Namespace oder ein Paket an:

1. In Diagrammen wird für Namespace-Elemente standardmäßig der Inhalt des Namespace angezeigt.
2. Sie können über das Kontextmenü einer Klasse bzw. eines Interface in einem Namespace direkt Felder und Methoden hinzufügen.

So öffnen Sie einen Namespace oder ein Paket:

1. Wählen Sie im Kontextmenü des Namespace-Elements den Befehl Diagramm öffnen.
2. Sie können auch auf das Namespace-Element im Diagramm doppelklicken.

So löschen Sie einen Namespace oder ein Paket:

1. Öffnen Sie die Diagrammansicht oder die Modellansicht.
2. Wählen Sie im Kontextmenü den Befehl Löschen.

Warnung: Durch das Löschen des Namespace wird sein gesamter Inhalt gelöscht.

So benennen Sie einen Namespace oder ein Paket um:

1. Öffnen Sie ein Projekt.
2. Führen Sie einen der folgenden Schritte aus, um den Namespace umzubenennen und den Namespace-Namen in allen zugehörigen Quelldateien zu ändern:
 - Wählen Sie in der Diagrammansicht oder der Modellansicht im Kontextmenü des Namespace den Befehl Umbenennen.
 - Öffnen Sie in der Diagrammansicht oder der Modellansicht den internen Editor für das Namespace-Element.
 - Bearbeiten Sie das Feld **Name** im Objektinspektor.

Siehe auch

Namespaces und Pakete – Überblick (siehe Seite 1446)

Ein Projekt erstellen (siehe Seite 249)

1.196 Beziehungen nach Pattern erstellen

Together bietet Ihnen die Möglichkeit, Beziehungen bequem per Pattern zu erstellen. Klicken Sie während der Modellierung in der Tool-Palette für das Diagramm auf die Schaltfläche Beziehung nach Pattern. Dadurch wird der Pattern-Experte mit einer Liste der verfügbaren Pattern geöffnet.

So erstellen Sie eine Beziehung nach Pattern:

1. Klicken Sie in der Tool-Palette für das Diagramm auf die Schaltfläche Beziehung nach Pattern. Die Schaltfläche wird gedrückt dargestellt.
2. Klicken Sie im Diagramm auf das Quellelement.
3. Ziehen Sie mit der Maus zum Zielement, und lassen Sie die Maustaste los, sobald das zweite Element optisch hervorgehoben wird. Der Pattern-Experte wird geöffnet.
4. Wählen Sie das Pattern für die neue Beziehung aus, legen Sie die Eigenschaften fest, und klicken Sie auf Fertig stellen.

Siehe auch

Überblick über Pattern (siehe Seite 1454)

Modellelemente nach Pattern erstellen (siehe Seite 227)

1.197 Modellelemente nach Pattern erstellen

Sie können ein Pattern anwenden, indem Sie in der Tool-Palette auf die Schaltfläche Knoten nach Pattern klicken oder in einem Kontextmenü den Befehl **Nach Pattern erstellen** wählen. Wenn Sie in einem Diagramm ein Element mit Hilfe einer Schaltfläche in der Symbolleiste erstellen, wird das Standard-Pattern angewendet, das der Schaltfläche zugewiesen ist.

So erstellen Sie Modellelemente nach Pattern:

1. Klicken Sie in der Tool Palette des Diagramms auf die Schaltfläche Knoten nach Pattern.
2. Klicken Sie auf den Container, dem Sie das Element hinzufügen möchten. Dies kann der Diagrammhintergrund oder ein Knotenelement sein. Der Pattern-Experte wird geöffnet.

Tip: Sie können auch mit der rechten Maustaste auf den Zielcontainer klicken und Nach Pattern erstellen wählen.

3. Wählen Sie das gewünschte Pattern im Pattern-Experten aus, ändern Sie die Eigenschaften nach Bedarf, und klicken Sie auf OK.

Siehe auch

Überblick über Pattern (siehe Seite 1454)

Beziehungen nach Pattern erstellen (siehe Seite 226)

1.198 Stub-Implementierungs-Pattern verwenden

So erstellen Sie mit der Schaltfläche Beziehung nach Pattern eine Vererbungsbeziehung mit Stub-Implementierung:

1. Klicken Sie in der Tool-Palette auf die Schaltfläche **Beziehung nach Pattern**.
2. Klicken Sie auf die Quellklasse, und ziehen Sie die Beziehung per Drag&Drop zur Zielklasse oder zum Ziel-Interface. Der Pattern-Experte wird geöffnet.
3. Erweitern Sie im Pattern-Experten den Ordner *Standard*, und wählen Sie **Implementation link and stub**.
4. Klicken Sie auf **OK**, um die Stub-Implementierung fertig zu stellen. Die Vererbungsbeziehung wird erstellt, und in der Quellklasse werden die Stubs für die geerbten Methoden erzeugt.

So erstellen Sie mit der Schaltfläche Knoten nach Pattern eine Vererbungsbeziehung mit Stub-Implementierung:

1. Klicken Sie in der Tool-Palette auf die Schaltfläche **Knoten nach Pattern**.
2. Wählen Sie die Quellklasse im Diagramm aus. Der Pattern-Experte wird geöffnet.
3. Erweitern Sie im Pattern-Experten den Ordner *Standard*, und wählen Sie **Implementation link and stub**.
4. Auf der rechten Seite des Pattern-Experten befindet sich der Bereich **Pattern-Eigenschaften**. Klicken Sie hier auf die Informationsschaltfläche rechts neben dem Feld **Anbieter**. Das Dialogfeld zur Auswahl eines Anbieters wird geöffnet.
5. Wählen Sie die Zielklasse bzw. das Ziel-Interface in der hierarchischen Anzeige mit den verfügbaren Elementen aus, und klicken Sie auf **OK**.
6. Klicken Sie auf **OK**, um die Stub-Implementierung fertig zu stellen und den Pattern-Experten zu beenden. Die Vererbungsbeziehung wird erstellt, und in der Quellklasse werden die Stubs für die geerbten Methoden erzeugt.

So erstellen Sie mit dem Kontextmenübefehl Nach Pattern erstellen eine Vererbungsbeziehung mit Stub-Implementierung:

1. Klicken Sie im Diagramm mit der rechten Maustaste auf die Quellklasse, und wählen Sie **Nach Pattern erstellen**. Der Pattern-Experte wird geöffnet.
2. Erweitern Sie im Pattern-Experten den Ordner *Standard*, und wählen Sie **Implementation link and stub**.
3. Auf der rechten Seite des Pattern-Experten befindet sich der Bereich **Pattern-Eigenschaften**. Klicken Sie hier auf die Informationsschaltfläche rechts neben dem Feld **Anbieter**. Das Dialogfeld zur Auswahl eines Anbieters wird geöffnet.
4. Wählen Sie die Zielklasse bzw. das Ziel-Interface in der hierarchischen Anzeige mit den verfügbaren Elementen aus, und klicken Sie auf **OK**.
5. Klicken Sie auf **OK**, um die Stub-Implementierung fertig zu stellen und den Pattern-Experten zu beenden. Die Vererbungsbeziehung wird erstellt, und in der Quellklasse werden die Stubs für die geerbten Methoden erzeugt.

Anmerkung: Das Stub-Implementierungs-Pattern wird im Kontextmenü von Klassen angeboten, die von einem Interface oder einer abstrakten Klasse erben. Außerdem ist dieses Pattern über die Schaltfläche **Knoten nach Pattern** in der Tool-Palette des Pattern-Experten und über den Befehl **Nach Pattern erstellen** im Kontextmenü von Klassen verfügbar. Verwenden Sie das Stub-Implementierungs-Pattern, um die Methoden in die Quellklasse zu kopieren, wenn im Diagramm bereits eine Vererbungs-/Generalisierungsbeziehung definiert ist.

So erstellen Sie eine Stub-Implementierung mit Hilfe des Kontextmenüs einer Klasse:

1. Klicken Sie mit der rechten Maustaste auf eine Klasse, die von einem Interface oder einer abstrakten Klasse erbt.
2. Klicken Sie im Kontextmenü auf **Stub-Implementierung**.

So erstellen Sie eine Stub-Implementierung mit der Schaltfläche Knoten nach Pattern:

1. Klicken Sie in der Tool-Palette auf die Schaltfläche **Knoten nach Pattern**.

2. Wählen Sie die Quellklasse im Diagramm aus. Der Pattern-Experte wird geöffnet.
3. Erweitern Sie im Pattern-Experten den Ordner *Standard*, und wählen Sie **Stub-Implementierung**.
4. Klicken Sie auf **OK**, um die Stub-Implementierung fertig zu stellen und den Pattern-Experten zu beenden. Die Stubs für die geerbten Methoden werden in der Quellklasse erzeugt.

So erstellen Sie eine Stub-Implementierung mit dem Kontextmenübefehl Nach Pattern erstellen:

1. Klicken Sie im Diagramm mit der rechten Maustaste auf die Quellklasse, und wählen Sie **Nach Pattern erstellen**. Der Pattern-Experte wird geöffnet.
2. Erweitern Sie im Pattern-Experten den Ordner *Standard*, und wählen Sie **Stub-Implementierung**.
3. Klicken Sie auf **OK**, um die Stub-Implementierung fertig zu stellen und den Pattern-Experten zu beenden. Die Stubs für die geerbten Methoden werden in der Quellklasse erzeugt.

Siehe auch

Überblick über Pattern ( siehe Seite 1454)

Pattern-Organizer ( siehe Seite 1271)

Pattern-Registrierung ( siehe Seite 1272)

1.199 Hinzufügen von Teilnehmern zu FCC-Pattern

Pattern als First-Class-Citizens werden durch die **GoF**-Pattern repräsentiert. Wenn Sie diese Pattern anwenden, werden die Elemente mit einer vorgegebenen Anzahl von Teilnehmern erstellt. Sie können einem vorhandenen Pattern-Objekt aber zulässige Teilnehmer hinzufügen. Dadurch werden Beziehungen zwischen dem Pattern-Objekt und den neuen Teilnehmern erstellt.

So fügen Sie einem GoF-Pattern einen Teilnehmer hinzu:

1. Wählen Sie das ovale Pattern-Element in der Diagrammansicht oder Modellansicht aus.
2. Klicken Sie mit der rechten Maustaste auf das Pattern-Element, und wählen Sie Hinzufügen. Ein Untermenü mit einer Liste der zulässigen Teilnehmer wird angezeigt.
3. Wählen Sie den gewünschten Teilnehmer aus.
4. Geben Sie im Experten für Pattern-Aktionen den Namen des neuen Teilnehmers ein, und klicken Sie auf OK.

Tip: Ist bereits ein Teilnehmer mit dem angegebenen Namen vorhanden, wird er verwendet.

Siehe auch

Überblick über Pattern (siehe Seite 1454)

Pattern-Organizer (siehe Seite 1271)

Pattern-Registrierung (siehe Seite 1272)

1.200 Ein Pattern erstellen

Sie können neue benutzerdefinierte Pattern auf der Grundlage vorhandener Diagrammelemente erstellen. Die neuen Pattern werden in der Pattern-Registrierung gespeichert und danach in der Pattern-Hierarchie des Pattern-Organizers angezeigt. Sie stehen dann für die Erstellung neuer Designelemente in Diagrammen zur Verfügung.

So erstellen Sie ein Pattern:

1. Wählen Sie ein oder mehrere Elemente in einem Diagramm aus.
2. Klicken Sie mit der rechten Maustaste, und wählen Sie im Kontextmenü Als Pattern speichern. Der Experte zum Erstellen neuer Pattern wird geöffnet.
3. Geben Sie auf der Seite Allgemein des Experten folgende Informationen ein:
 - Geben Sie in das Feld Datei den Namen der XML-Zieldatei ein.
 - Geben Sie in das Feld Name den Namen für das neue Pattern ein.
 - Geben Sie in das Feld Beschreibung eine Beschreibung für das Pattern ein (optional).
 - Aktivieren Sie gegebenenfalls das Kontrollkästchen Pattern-Objekt erzeugen. Das Pattern kann dann als First-Class-Citizen fungieren. Wenn Sie das Pattern anwenden, wird im Diagramm ein ovales Pattern-Element angezeigt.
 - Klicken Sie auf Weiter.
4. Auf der Seite Pattern-Parameter des Experten haben Sie folgende Möglichkeiten:
 - Ändern Sie die Parameter im integrierten Editor nach Bedarf.
 - Aktivieren Sie gegebenenfalls die Einstellung Vorhandene benutzen für das Pattern. Wenn diese Einstellung aktiv ist, werden vorhandene Elemente im Diagramm bei der Anwendung des Pattern wieder verwendet. Befindet sich im Zielcontainer bereits ein Element mit dem gleichen Namen und Metatyp, wird bei der Anwendung des Pattern kein neues Element erstellt. Sollen neue Elemente erstellt werden, deaktivieren Sie die Einstellung Vorhandene benutzen.
 - Klicken Sie auf Weiter.
5. Die Seite Hierarchieordner auswählen mit der aktuellen Pattern-Struktur wird angezeigt. Wählen Sie den Zielordner aus, und klicken Sie auf OK.

Ergebnis: Das neue Pattern wird im angegebenen Ordner gespeichert. Das Pattern wird nun in der Pattern-Hierarchie aufgeführt und kann zur Erstellung neuer Designelemente verwendet werden.

Siehe auch

Überblick über Pattern (siehe Seite 1454)

Pattern erstellen (Experte) (siehe Seite 1278)

Pattern-Organizer (siehe Seite 1271)

Pattern-Registrierung (siehe Seite 1272)

1.201 Entfernen von FCC-Pattern aus dem Modell

Sie können Elemente von Pattern als First-Class-Citizens (GoF-Pattern) sowohl in der Diagrammansicht als auch in der Modellansicht löschen. Die gelöschten Elemente werden aus dem Diagramm und dem Modell entfernt.

So löschen Sie ein GoF-Pattern mit Teilnehmern:

1. Markieren Sie das zu löschen ovale Pattern-Element in der Diagrammansicht oder Modellansicht.
2. Klicken Sie mit der rechten Maustaste auf das Element, und wählen Sie Mit Teilnehmern löschen.
3. Bestätigen Sie die Löschung.

Siehe auch

Überblick über Pattern (siehe Seite 1454)

Pattern-Organizer (siehe Seite 1271)

Pattern-Registrierung (siehe Seite 1272)

1.202 Pattern exportieren

Sie können ein Pattern nach der Erstellung in einen bestimmten Ordner exportieren.

So exportieren Sie ein Pattern:

1. Erweitern Sie im Pattern-Organizer die Pattern-Hierarchie, und markieren Sie den Ordner, der exportiert werden soll.
2. Klicken Sie mit der rechten Maustaste auf den Ordner, und wählen Sie Ordner exportieren.
3. Navigieren Sie im Dialogfeld Exportpfad auswählen zum gewünschten Speicherort, und klicken Sie auf Speichern.

Siehe auch

[Ein Pattern erstellen](#) (siehe Seite 231)

[Pattern-Organizer](#) (siehe Seite 1271) [Pattern-Registrierung](#) (siehe Seite 1272)

1.203 Vorhandene Pattern importieren

Sie können Pattern, die mit anderen Versionen von Together erzeugt wurden, importieren und wieder verwenden. Beim Start von Together werden die verfügbaren Speicherorte nach Pattern durchsucht und alle gefundenen Pattern in die Pattern-Registrierung aufgenommen. Die Pattern können aber erst verwendet werden, nachdem für sie im Pattern-Organizer Verknüpfungen erstellt wurden.

Damit ein benutzerdefiniertes Pattern wieder verwendet werden kann, müssen folgende Schritte ausgeführt werden:

1. Kopieren Sie die vorhandenen Pattern in den Pattern-Ordner unterhalb des Produktinstallationsordner.
2. Nach dem Start des Programms werden alle verfügbaren Pattern in der Pattern-Registrierung registriert.
3. Öffnen Sie den Pattern-Organizer,
4. und führen Sie folgende Schritte aus:
 - Geben Sie den Zielordner für die Pattern an, oder erstellen Sie einen neuen Ordner.
 - Erstellen Sie eine neue Verknüpfung.
 - Weisen Sie der Verknüpfung das gewünschte Pattern zu.
5. Speichern Sie die Änderungen.

Siehe auch

Überblick über Pattern ([siehe Seite 1454](#))

Neuen Ordner erstellen ([siehe Seite 238](#))

Neue Verknüpfung erstellen ([siehe Seite 238](#))

Pattern an eine Verknüpfung zuweisen ([siehe Seite 236](#))

Änderungen im Pattern-Organizer speichern ([siehe Seite 244](#))

Pattern-Organizer ([siehe Seite 1271](#)) Pattern-Registrierung ([siehe Seite 1272](#))

1.204 Gemeinsame Nutzung von Pattern

Um die Entwicklung im Team zu erleichtern, können Pattern an gemeinsam genutzten Speicherorten bereitgestellt werden. Nachdem die Pfade der Pattern in die Liste der Verzeichnisse der gemeinsam genutzten Pattern aufgenommen wurden, kann über den Pattern-Organizer darauf zugegriffen werden. Alle Pattern, die in dieser Liste enthalten sind, werden im Knoten für benutzerdefinierte Pattern der Pattern-Hierarchie angezeigt.

So erstellen Sie gemeinsam genutzte Pattern:

1. Exportieren Sie die Pattern in ein gemeinsam genutztes Verzeichnis.
2. Klicken Sie im Pattern-Organizer auf Verzeichnisse der gemeinsam genutzten Pattern bearbeiten. Das Dialogfeld Verzeichnisse der gemeinsam genutzten Pattern wird angezeigt.
3. Klicken Sie in der Liste der Verzeichnisse mit gemeinsam genutzten Pattern auf Hinzufügen. Das Dialogfeld Gemeinsame Pattern-Hierarchie auswählen wird angezeigt.
4. Navigieren Sie in diesem Dialogfeld zu dem Ordner mit den gewünschten Pattern, wählen Sie die Datei `Shortcut Registry.xml` aus, und klicken Sie auf Öffnen. Der Pfad wird der Liste mit den Verzeichnissen der gemeinsam genutzten Pattern hinzugefügt.
5. Bearbeiten Sie die Liste mit den Schaltflächen Hinzufügen und Entfernen.
6. Klicken Sie auf OK, um den Vorgang abzuschließen.

Siehe auch

Pattern exportieren ( siehe Seite 233)

Pattern-Organizer ( siehe Seite 1271)

Pattern-Registrierung ( siehe Seite 1272)

1.205 Pattern an Verknüpfungen zuweisen

Sie können ein Pattern einer oder mehreren Verknüpfungen zuweisen, die sich in verschiedenen virtuellen Ordnern befinden.

So weisen Sie einer Verknüpfung ein Pattern zu:

1. Wählen Sie im Bereich Virtuelle Pattern-Hierarchien des Pattern-Organizers die gewünschte Verknüpfung aus.
2. Klicken Sie mit der rechten Maustaste, und wählen Sie Pattern zuweisen. Die Pattern-Registrierung wird geöffnet.
3. Wählen Sie in der Pattern-Registrierung das Pattern aus, das der gewählten Verknüpfung zugewiesen werden soll, und klicken Sie auf OK.
4. Legen Sie im Bereich Eigenschaften des Pattern-Organizers die Eigenschaft für den Namen und die Sichtbarkeit nach Bedarf fest.
5. Speichern Sie die Änderungen.

Siehe auch

Verknüpfung zu einem Pattern erstellen ([siehe Seite 239](#)) Pattern-Eigenschaften bearbeiten ([siehe Seite 242](#))

Pattern-Organizer ([siehe Seite 1271](#))

Pattern-Registrierung ([siehe Seite 1272](#))

1.206 Verknüpfungen, Ordner und Pattern-Hierarchien kopieren und einfügen

So können Sie einen Ordner, eine Verknüpfung oder eine Pattern-Hierarchie kopieren und einfügen:

1. Wählen Sie die zu kopierende Verknüpfung, den Ordner oder die Pattern-Hierarchie im Bereich Virtuelle Pattern-Hierarchien aus.
2. Klicken Sie mit der rechten Maustaste auf den Knoten, und wählen Sie Kopieren.
3. Klicken Sie mit der rechten Maustaste auf den Zielknoten, und wählen Sie Einfügen. Drücken Sie alternativ STRG+V.
4. Speichern Sie die Änderungen.

Siehe auch

Pattern-Organizer (siehe Seite 1271)

Pattern-Registrierung (siehe Seite 1272)

1.207 Ordner erstellen

Mit Hilfe von virtuellen Ordnern können Sie die Pattern in den Pattern-Hierarchien logisch anordnen.

So erstellen Sie einen Ordner:

1. Wählen Sie im Pattern-Organizer den Zielknoten im Bereich Virtuelle Pattern-Hierarchien aus.
2. Klicken Sie mit der rechten Maustaste auf den Knoten, und wählen Sie Neuer Ordner. Der Knoten *Neuer Ordner* wird hinzugefügt.
3. Bearbeiten Sie im Bereich Eigenschaften die Felder Name und Sichtbar nach Bedarf.

Siehe auch

Verknüpfung zu einem Pattern erstellen ( siehe Seite 239) Hierarchie erstellen ( siehe Seite 240)

Pattern-Organizer ( siehe Seite 1271)

Pattern-Registrierung ( siehe Seite 1272)

1.208 Verknüpfung zu einem Pattern erstellen

Im Pattern-Organizer arbeiten Sie nicht mit den Pattern selbst, sondern mit ihren Verknüpfungen. Daher können Verknüpfungen mit demselben Pattern in mehreren Ordnern vorhanden sein.

So erstellen Sie eine Verknüpfung zu einem Pattern:

1. Wählen Sie im Pattern-Organizer den obersten Zielknoten aus.
2. Klicken Sie mit der rechten Maustaste auf den Knoten, und wählen Sie Neue Verknüpfung. Die Pattern-Registrierung wird geöffnet.
3. Wählen Sie in der Pattern-Registrierung das Pattern aus, das der neuen Verknüpfung zugewiesen werden soll, und klicken Sie auf OK.
4. Der Pattern-Organizer fordert Sie auf, die Änderungen in der Pattern-Registrierung zu speichern. Klicken Sie auf Ja.

Siehe auch

Ordner erstellen (siehe Seite 238) Hierarchie erstellen (siehe Seite 240)

Pattern-Organizer (siehe Seite 1271)

1.209 Virtuelle Pattern-Hierarchie erstellen

Im Pattern-Organizer können Sie Pattern unter Verwendung von virtuellen Hierarchien, Ordnern und Verknüpfungen logisch organisieren. Unterhalb eines Hierarchieknotens können Sie virtuelle Ordner und Verknüpfungen zu Pattern erstellen.

So erstellen Sie eine Pattern-Hierarchie:

1. Wählen Sie im Pattern-Organizer den Pattern-Knoten auf der obersten Ebene aus.
2. Klicken Sie mit der rechten Maustaste auf den Knoten, und wählen Sie Neue Pattern-Hierarchie. Der Knoten *Neue Pattern-Hierarchie* wird hinzugefügt.
3. Bearbeiten Sie im Bereich Eigenschaften die Felder Name und Sichtbar nach Bedarf.

Siehe auch

Ordner erstellen (siehe Seite 238) Verknüpfung zu einem Pattern erstellen (siehe Seite 239)

Pattern-Organizer (siehe Seite 1271)

1.210 Verknüpfungen, Ordner und Pattern-Hierarchien löschen

So löschen Sie einen Knoten im Pattern-Organizer:

1. Wählen Sie die Verknüpfung, den Ordner oder die Pattern-Hierarchie im Bereich Virtuelle Pattern-Hierarchien aus.
2. Klicken Sie mit der rechten Maustaste auf den Knoten, und wählen Sie Löschen. Drücken Sie alternativ die Taste ENTF.
3. Speichern Sie die Änderungen.

Siehe auch

Pattern-Organizer (siehe Seite 1271)

Pattern-Registrierung (siehe Seite 1272)

1.211 Eigenschaften bearbeiten

Die Eigenschaften von virtuellen Hierarchien, Ordnern und Verknüpfungen werden im Bereich **Eigenschaften** des Pattern-Organizers angezeigt. Sie können die Darstellung der Eigenschaften mit Hilfe von Schaltflächen in der Symbolleiste ändern. Die Eigenschaften können in erweiterbaren Knoten oder in alphabetischer Reihenfolge angezeigt werden. Die Eigenschaften Name und Sichtbar können bearbeitet werden. Die Änderungen werden wirksam, sobald das bearbeitete Feld den Fokus verliert oder die EINGABETASTE gedrückt wird. Der Knotenname in der Hierarchieansicht wird entsprechend angepasst.

So bearbeiten Sie die Eigenschaften einer Hierarchie, einer Verknüpfung oder eines Ordners:

1. Wählen Sie einen Knoten im Bereich Virtuelle Pattern-Hierarchien aus.
2. Nehmen Sie im Bereich Eigenschaften die gewünschten Änderungen im Textfeld für die Eigenschaft Name vor.
3. Um die Eigenschaft Sichtbar zu ändern, wählen Sie den gewünschten Wert aus der Dropdown-Liste.

Tip: Die Eigenschaft Sichtbar ist nur für Verknüpfungen verfügbar. Wenn sie auf *Visible* gesetzt ist, wird die Verknüpfung im Pattern-Experten angezeigt. Andernfalls ist sie ausgeblendet. Ordner, die keine sichtbaren Verknüpfungen enthalten, werden im Pattern-Experten ebenfalls ausgeblendet.

4. Speichern Sie die Änderungen.

Siehe auch

Pattern-Organizer (siehe Seite 1271)

Pattern-Registrierung (siehe Seite 1272)

1.212 Pattern-Organizer öffnen

Mit Hilfe des Pattern-Organizers können Sie Pattern logisch organisieren (mit virtuellen Hierarchien, Ordnern und Verknüpfungen) und die Eigenschaften von Pattern anzeigen und bearbeiten.

So öffnen Sie den Pattern-Organizer:

1. Wählen Sie im Hauptmenü **Tools** ▶ **Pattern-Organizer**.
2. Ergebnis: Das Dialogfeld Pattern-Organizer wird geöffnet.

Siehe auch

Pattern-Organizer (↗ siehe Seite 1271)

Pattern-Registrierung (↗ siehe Seite 1272)

1.213 Änderungen in der Pattern-Registrierung speichern

Wenn Sie Änderungen im Pattern-Organizer vornehmen (Definieren neuer Verknüpfungen, Exportieren oder Erstellen gemeinsam genutzter Ordner usw.), wird der Inhalt der Pattern-Registrierung automatisch synchronisiert. Beim Schließen des Pattern-Organizers werden Sie aufgefordert, die Änderungen zu speichern. Beim jedem Start von Together werden die verfügbaren Speicherorte nach Pattern durchsucht. Der Inhalt der Registrierung wird synchronisiert und entspricht den tatsächlich verfügbaren Pattern-Ordnern. Änderungen an Pattern, die Sie außerhalb des Organizers vornehmen, werden beim Start von Together synchronisiert.

So speichern Sie Änderungen in der Pattern-Registrierung:

1. Klicken Sie im Pattern-Organizer auf die Schaltfläche Schließen. Sie werden in einem Dialogfeld aufgefordert, die Änderungen in der Pattern-Registrierung zu speichern.
2. Bestätigen Sie mit Ja.

Tip: Sie können die Synchronisierung auch durchführen, indem Sie im Dialogfeld Pattern-Registrierung auf Synchronisieren klicken.

Siehe auch

Pattern-Organizer (siehe Seite 1271)

Pattern-Registrierung (siehe Seite 1272)

1.214 Pattern sortieren

Logische Hierarchien, Ordner und Verknüpfungen werden im Pattern-Organizer möglicherweise in einer willkürlichen Reihenfolge angezeigt. Mit dem Befehl Ordner sortieren können Sie Knoten innerhalb eines Containerknotens alphabetisch sortieren.

So sortieren Sie Pattern im Pattern-Organizer:

1. Wählen Sie den zu sortierenden Knoten im Bereich Virtuelle Pattern-Hierarchien aus.
2. Klicken Sie mit der rechten Maustaste auf den Knoten, und wählen Sie Ordner sortieren.
3. Speichern Sie die Änderungen.

Siehe auch

Pattern-Organizer (siehe Seite 1271)

Pattern-Registrierung (siehe Seite 1272)

1.215 Den Pattern-Organizer verwenden

Mit dem Pattern-Organizer können Sie folgende Aktionen durchführen:

- Logische Pattern-Hierarchien und Ordner erstellen
- Verknüpfungen zu Pattern erstellen
- Pattern an Verknüpfungen zuweisen
- Hierarchien, Ordner und Verknüpfungen kopieren, ausschneiden und löschen
- Änderungen in der Pattern-Registrierung speichern

Siehe auch

Überblick über Pattern ([siehe Seite 1454](#))

Pattern-Organizer öffnen ([siehe Seite 243](#))Ordner erstellen ([siehe Seite 238](#))Virtuelle Pattern-Hierarchie erstellen ([siehe Seite 240](#))Verknüpfung zu einem Pattern erstellen ([siehe Seite 239](#))Pattern an Verknüpfungen zuweisen ([siehe Seite 236](#))Im Pattern-Organizer kopieren und einfügen ([siehe Seite 237](#))Knoten im Pattern-Organizer löschen ([siehe Seite 241](#))Pattern-Eigenschaften bearbeiten ([siehe Seite 242](#))Im Pattern-Organizer sortieren ([siehe Seite 245](#))Änderungen im Pattern-Organizer speichern ([siehe Seite 244](#))

Pattern-Organizer ([siehe Seite 1271](#))Pattern-Registrierung ([siehe Seite 1272](#))

1.216 Die Pattern-Registrierung verwenden

Die Pattern-Registrierung kann über das Kontext-Menü des Pattern-Organizers geöffnet werden, wenn eine neue Verknüpfung erstellt oder einer Verknüpfung ein Pattern zugewiesen wird. In der Pattern-Registrierung können Pattern nach Kategorie, Metaklasse, Diagrammtyp, Sprache und Registrierungsstatus gefiltert werden.

Sie können die Pattern-Registrierung auf folgende Arten öffnen:

- Klicken Sie mit der rechten Maustaste auf einen Ordner, und wählen Sie Neue Verknüpfung.
- Klicken Sie mit der rechten Maustaste auf eine Pattern-Verknüpfung, und wählen Sie Pattern zuweisen.

So filtern Sie Pattern in der Pattern-Registrierung:

1. Klicken Sie im Bereich **Filter** des Dialogfelds **Pattern-Registrierung** auf das Attribut, nach dem die Pattern gefiltert werden sollen.
2. Wählen Sie den gewünschten Wert aus der Dropdown-Liste.

Siehe auch

Überblick über Pattern (siehe Seite 1454)

Pattern-Organizer (siehe Seite 1271)

Pattern-Registrierung (siehe Seite 1272)

1.217 Together-Unterstützung für Projekte aktivieren

In diesem Thema wird beschrieben, wie Sie die Together-Unterstützung für ein einzelnes Projekt aktivieren.

Tip: Über die allgemeinen Optionen können Sie die Together-Unterstützung für alle neuen oder aktuell geöffneten Projekte aktivieren.

So aktivieren Sie die Together-Unterstützung:

1. Wechseln Sie zum gewünschten Projekt bzw. zur gewünschten Projektgruppe.
2. Wählen Sie im Hauptmenü **Projekt** ▶ **Together-Unterstützung**.

Tip: Sie können stattdessen auch Together-Unterstützung im Kontextmenü des Projektknotens in der Projektverwaltung oder in der Strukturansicht wählen.

Ergebnis: Das Dialogfeld Modellunterstützung wird geöffnet. Es enthält eine Liste der Projekte in der aktuellen Projektgruppe.

3. Markieren Sie im Dialogfeld Modellunterstützung die Projekte, für die Modellierungsfunktionen benötigt werden.
4. Klicken Sie auf OK.

Ergebnis: Die Modelle für die gewählten Projekte werden in der Modellansicht angezeigt. In der Projektverwaltung wird jedem gewählten Projekt der Ordner ModelSupport_%PROJEKTNAMEN%ModelSupport hinzugefügt.

Wenn Sie die Together-Unterstützung wieder deaktivieren möchten, wiederholen Sie die obigen Schritte und entfernen die Markierung von den Projekten der Projektgruppe, für die keine Modellierungsfunktionen benötigt werden.

Siehe auch

Ein Projekt erstellen (siehe Seite 249)

Vorhandenes Projekt für die Modellierung öffnen (siehe Seite 257)

Allgemeine Optionen von Together (siehe Seite 1281)

1.218 Ein Projekt erstellen

So erstellen Sie ein Together-Projekt:

1. Wählen Sie im Hauptmenü **Datei > Neu > Weitere**. Das Dialogfeld Objektgalerie wird geöffnet.
2. Wählen Sie im Bereich Projekttypen oder Elementkategorien die gewünschte Projektkategorie aus.
3. Wählen Sie im Templates-Bereich eine Projekt-Template aus.
4. Geben Sie im Dialogfeld Neue Anwendung den Projektnamen, den Speicherort und die anderen erforderlichen Parameter an.
5. Klicken Sie auf OK.
6. Folgen Sie den Anleitungen des Experten für neue Projekte.
7. Klicken Sie im Experten zum Erstellen eines Projekts aus MDL auf die Schaltfläche **Ordner hinzufügen**, und wählen Sie den gewünschten Quellordner im Dateisystem aus. Mit den Schaltflächen **Entfernen** und **Alle entfernen** können Sie nicht benötigte Modelldateien aus der Liste entfernen.

Ergebnis: Im angegebenen Verzeichnis wird ein neues Projekt des angegebenen Typs erstellt.

Bei einem Designprojekt wird im angegebenen Stammordner eine **.bdsproj.tgproj**-Datei erstellt. Außerdem werden das Standardpaket und das Standarddiagramm erstellt.

Bei einem Implementierungsprojekt werden bei aktiverer Together-Unterstützung eine **.bdsproj**-Datei im angegebenen Stammordner, der Standard-Namespace und das Standarddiagramm erstellt.

Siehe auch

Projekt für die Modellierung öffnen ( siehe Seite 257)

Namespace oder Paket erstellen ( siehe Seite 225)

Diagramme erstellen ( siehe Seite 116)

Unterstützte Projektformate ( siehe Seite 1332)

1.219 Projekt in das XMI-Format exportieren

So exportieren Sie ein Projekt in das XMI-Format:

1. Klicken Sie in der Modellansicht mit der rechten Maustaste auf den Stammknoten des Projekts, und wählen Sie **Projekt nach XMI exportieren**. Sie können stattdessen auch **Datei > Projekt nach XMI exportieren** im Hauptmenü wählen. Das Dialogfeld XMI-Export wird geöffnet.
2. Wählen Sie im Gruppenfeld **XMI-Typ auswählen** die XML-/UML-Version für die Datei aus. Folgende XMI-Typen stehen zur Verfügung:
 - XMI für UML 1.3 (Unisys-Erweiterung)
 - XMI für UML 1.3 (Unisys-Erweiterung, empfohlen für TCC), Standardeinstellung
 - XMI für UML 1.3 (Unisys-Erweiterung, empfohlen für IBM Rational Rose)
3. Klicken Sie auf den Pfeil der Dropdown-Liste, um die entsprechende XMI-Codierung auszuwählen. Der Standardwert ist **UTF-8**.
4. Legen Sie das Ziel des Exports fest. Sie können sowohl den Pfad als auch den Namen der Datei (.xml) angeben, die erstellt wird. Oder Sie übernehmen die Standardeinstellung: (**Projektordner**)\out\xmi\(**Projektname**).xml
5. Klicken Sie auf **Exportieren**. Wenn das Zielverzeichnis nicht vorhanden ist, müssen Sie die Erstellung dieses Verzeichnisses in einem Dialogfeld bestätigen.
6. Klicken Sie auf **Ja**.

Ergebnis: Die erzeugte XML-Datei wird an der angegebenen Position gespeichert.

Siehe auch

Überblick zum Importieren und Exportieren (siehe Seite 1459)

XMI-Export (Dialogfeld) (siehe Seite 710)

1.220 Projekt im IBM Rational Rose-Format (MDL) importieren

So erstellen Sie ein Designprojekt aus einem vorhandenen IBM Rational Rose-Projekt (MDL):

1. Wählen Sie im Hauptmenü **Datei > Neu > Weitere**. Das Dialogfeld Neues Projekt wird geöffnet.
2. Wählen Sie im Bereich Projekttypen die Option **Designprojekt**.
3. Wählen Sie im Bereich Templates die Template Aus MDL konvertieren.
4. Geben Sie im Dialogfeld Neues Projekt den Projektnamen, den Speicherort und die anderen erforderlichen Parameter an.
5. Klicken Sie auf OK.
6. Klicken Sie im Experten zum Erstellen eines Projekts aus MDL auf die Schaltfläche Hinzufügen, und wählen Sie die **.mdl-, .ptl-, .cat- oder .sub-Datei** aus.
7. Legen Sie den Skalierungsfaktor und die Konvertierungsoptionen fest.
8. Klicken Sie auf Fertig stellen.

Ergebnis: Im angegebenen Verzeichnis wird ein neues Designprojekt erstellt.

Siehe auch

Überblick zum Importieren und Exportieren (siehe Seite 1459)

Konvertieren aus MDL (Experte) (siehe Seite 1276)

1.221 In TCC oder TAR erstellte Projekte importieren

Nachfolgend wird beschrieben, wie Sie Together ControlCenter-Diagramme und -Quelltext in RAD Studio verwenden können.

Damit die Together ControlCenter-Modelle in RAD Studio verwendet werden können, unterstützt Together ControlCenter die Paket-Namespace-Organisation für C#-Projekte. Sie können bei Bedarf im Dialogfeld Optionen von Together ControlCenter das automatische Zuweisen der Namespaces für C#-Klassifizierer festlegen. Wenn Sie eine Klasse einem Paket hinzufügen (unabhängig davon, wo sich die Klasse vorher befunden hat), wird ihre neue Instanz in die Namespace-Deklaration eingefügt, die dem Paketnamen entspricht. Der Name des Namespace ist mit dem des Pakets identisch, und den in der Verzeichnis-/Paketstruktur platzierten Dateien werden die betreffenden Namespaces im Quelltext zugewiesen.

Warnung: Sie können keine Pakete verschieben, da sonst die Namespace-Organisation verletzt wird. Wenn Sie ein Paket verlagern möchten, müssen Sie ein neues Paket mit demselben Namen im betreffenden Ordner erstellen, danach im neuen Paket die gewünschten Klassen erstellen und zuletzt das alte Paket entfernen.

Anmerkung: Dieses Beispiel setzt Folgendes voraus:

Sie haben die Together-Unterstützung bereits installiert und aktiviert.

Sie haben bereits mit Ihren C#-Projekten in Together ControlCenter 6.2 oder Together Architect bei aktiver Paket-Namespace-Organisation gearbeitet (siehe unten), sodass sich Ihre Quelltextdateien in den entsprechenden Namespaces in RAD Studio befinden.

Das Importieren eines Objekts umfasst folgende Schritte:

1. Unterstützung für die Paket-Namespace-Organisation aktivieren.
2. Das Projekt in RAD Studio einrichten.
- 3.
4. Die Together ControlCenter-Dateistruktur dem RAD Studio-Projekt hinzufügen.
5. Die Together ControlCenter-Projektdateien in RAD Studio anzeigen.

So konfigurieren Sie TCC oder TAR für die automatische Namespace-Zuweisung bei Klassifizierern in Implementierungsprojekten:

1. Öffnen Sie das Projekt in TCC oder TAR.
2. Wählen Sie **Tools**▶**Optionen**▶**Standardebene** oder **Tools**▶**Optionen**▶**Projektebene**.
3. Erweitern Sie im linken Bereich des Dialogfelds Optionen den Knoten **Quelltext**▶**C#**.
4. Aktivieren Sie die Option für die Namespace-Zuweisung nach Paketnamen.
5. Klicken Sie auf OK, um die Einstellungen zu übernehmen und das Dialogfeld zu schließen.
6. Schließen Sie das TCC- bzw. TAR-Projekt.

So richten Sie ein Projekt in Delphi ein:

1. Wählen Sie **Datei**▶**Neu**▶**Weitere**. Das Dialogfeld Neues Projekt wird geöffnet.
2. Wählen Sie als Projekttyp **C#-Projekte**
3. Wählen Sie im Bereich **Templates** die gewünschte Template aus.

4. Klicken Sie auf OK.
 5. Geben Sie einen Namen für das Projekt ein. Verwenden Sie den Namen des TCC- bzw. TAR-Projekts.
 6. Geben Sie einen Speicherort für das Projekt an. Der tatsächliche Speicherort ist nicht von Bedeutung, Sie benötigen die Angabe aber später.
 7. Klicken Sie auf OK.
- Ergebnis: Das Projekt wird erstellt und in RAD Studio angezeigt.

So erstellen Sie die Dateistruktur:

1. Schließen Sie das Projekt in RAD Studio.
2. Öffnen Sie Windows-Explorer oder ein anderes Dateiverwaltungsfenster.
3. Wechseln Sie zum TCC- oder TAR-Projektordner.
4. Kopieren Sie den Ordner `src` des TCC- oder TAR-Projekts in das RAD Studio-Projekt.
5. Öffnen Sie den Ordner `diagrams` des TCC- bzw. TAR-Projekts, und kopieren Sie seinen Inhalt in den Ordner `ModelSupport_%PROJEKTNAMEN%ModelSupport` des RAD Studio-Projekts. In RAD Studio werden die Diagrammdateien standardmäßig im Ordner `ModelSupport_%PROJEKTNAMEN%ModelSupport` und in TCC (TAR) im Ordner `diagrams` gespeichert.

So fügen Sie die TCC- oder TAR-Quelltextelement dem neuen Projekt hinzu:

1. Wechseln Sie zu RAD Studio.
2. Klicken Sie im Hauptmenü auf **Datei > Neu öffnen**, und wählen Sie das neue Projekt aus der Liste.
3. Öffnen Sie die Projektverwaltung.
4. Wählen Sie den Stammknoten des Projekts aus.
5. Klicken Sie mit der rechten Maustaste, und wählen Sie Hinzufügen. Das Dialogfeld Dem Projekt hinzufügen wird angezeigt.
6. Wählen Sie in diesem Dialogfeld die erste Quelltextdatei im Ordner `src` aus, und klicken Sie auf OK.
7. Wiederholen Sie diese Schritte für alle Quelltext- und Modelldateien.
8. angezeigt wird, klicken Sie auf die Schaltfläche **Alle Dateien anzeigen** am oberen Rand der Projektverwaltung.
9. 10.
11. Öffnen Sie die Modellansicht.

Ergebnis: Die Diagramm- und Quelltextdateien des TCC- bzw. TAR-Projekts werden in der Modellansicht angezeigt.

Siehe auch

Überblick zur Interoperabilität (siehe Seite 1460)

Gemeinsame Nutzung von Modellinformationen in TCC/TAR und RAD Studio (siehe Seite 258)

1.222 In TVS, TEC, TJB oder TPT erstellte Projekte importieren

Together unterstützt eine hundertprozentige Abwärtskompatibilität mit früheren Versionen. Ältere Projekte können wie gewohnt geöffnet werden.

Außerdem können Sie Projekte, die in anderen Editionen von Together erstellt wurden, importieren.

Warnung: Diagramme in Projekten müssen im gemeinsamen Diagrammformat `.txv*` erstellt werden. Das alte Diagrammformat `.df*` wird nicht unterstützt.

Warnung: Die Diagrammelemente müssen eingebettet sein (als Dateimitglieder erstellt werden). Standalone-Designelemente (SDE) werden nicht unterstützt.

Der Import eines in TVS, TEC, TJB oder TPT erstellten Projekts umfasst folgende Schritte:

1. Ein neues Projekt in RAD Studio erstellen.
2. Die Modellinformationen in dieses Projekt importieren.

So erstellen Sie ein neues Projekt für den Import:

1. Wählen Sie im Hauptmenü **Datei**▶**Neu**▶**Weitere**. Das Dialogfeld Objektgalerie wird geöffnet.
 2. Wählen Sie die Projekt-Template aus. Beachten Sie, dass der Projekttyp dem Typ des Quelltextprojekts entsprechen muss:
 - Wählen Sie bei einem C#-Projekt **C#-Projekte**▶**(entsprechende Template)**.
- *
- Wählen Sie bei einem UML 1.x-Designprojekt **Designprojekt**▶**UML 1.5-Designprojekt**.
 - Wählen Sie bei einem UML 2.x-Designprojekt **Designprojekt**▶**UML 2.0-Designprojekt**.

3. Geben Sie einen Namen für das Projekt ein.

Warnung: Der Projektname muss mit dem Namen des Quelltextprojekts identisch sein. Legen Sie die restlichen Einstellungen fest.

4. Klicken Sie auf OK, um das Projekt zu erstellen.
5. Schließen Sie das Projekt, nachdem es erstellt wurde.

So importieren Sie die Modellinformationen:

1. Öffnen Sie Windows-Explorer oder ein anderes Dateiverwaltungsfenster.
2. Kopieren Sie alle Modelldateien einschließlich der Unterordner aus dem Quelltextprojekt in den Ordner `ModelSupport_%PROJEKTNAMEN%ModelSupport` im Stammordner des neuen Projekts. Diese Dateien befinden sich je nach Together-Version im Verzeichnis `diagrams`, `ModelSupport` oder `Model` Folder.

Anmerkung: Bei manchen Projekten befinden sich diese Dateien in denselben Ordnern wie die Quelltextdateien. Sie müssen dann manuell gesucht und kopiert werden. Sie benötigen alle Dateien mit der Namenserweiterung `.txv*` und `.txa*`.

3. Wenn Sie bei einem Implementierungsprojekt den Quelltext weiterhin verwenden möchten, kopieren Sie ihn aus dem Quelltextprojekt unter Beibehaltung der Ordnerstruktur in das neue Projekt.

4. Öffnen Sie das Projekt in RAD Studio. Öffnen Sie die Projektverwaltung.
5. Wählen Sie den Stammknoten des Projekts aus.
6. Klicken Sie mit der rechten Maustaste, und wählen Sie Hinzufügen. Das Dialogfeld Dem Projekt hinzufügen wird angezeigt. Wählen Sie in diesem Dialogfeld die erste Quelltextdatei im Ordner `src` aus, und klicken Sie auf OK.
7. Wiederholen Sie diese Schritte für alle Quelltext- und Modelldateien.
8. auf Alle Dateien anzeigen. Die neuen Dateien und Ordner werden dann im
9. Ergebnis: RAD Studio verarbeitet die Dateien. Im Anschluss wird das importierte Projekt in der Modellansicht und in der Diagrammansicht angezeigt.

Siehe auch

Überblick zur Interoperabilität (↗ siehe Seite 1460)

Gemeinsame Nutzung von Modellinformationen in TCC/TAR und RAD Studio (↗ siehe Seite 258)

1.223 Projekt im XMI-Format importieren

So importieren Sie ein Projekt im XMI-Format:

1. Öffnen Sie ein Diagramm, oder wählen Sie in der Modellansicht den Stammknoten des Projekts aus.

Warnung: Das Projekt muss mit der UML 1.5-Spezifikation konform sein.

2. Klicken Sie in der Modellansicht mit der rechten Maustaste auf den Stammknoten des Projekts, und wählen Sie Projekt aus XMI importieren. Sie können stattdessen auch **Datei>Projekt aus XMI importieren** im Hauptmenü wählen. Das Dialogfeld XMI-Import wird geöffnet.

3. Suchen Sie nach der Quelldatei.

4. Klicken Sie auf Importieren.

Tip: Verwenden Sie zum Importieren eines Projekts aus Together ControlCenter (TCC) oder Together Architect (TAR) in RAD Studio das gemeinsame Diagrammformat.

Modelle, die mit IBM Rational Rose erstellt wurden, können direkt importiert werden.

Siehe auch

Überblick zum Importieren und Exportieren (siehe Seite 1459)

Projekt im IBM Rational Rose-Format (MDL) importieren (siehe Seite 251)

1.224 Vorhandenes Projekt für die Modellierung öffnen

Sie können ein Implementierungsprojekt, das ohne Together erstellt wurde, mit Modellierungsfunktionen ausstatten.

Wenn Sie in der Modellansicht oder Diagrammansicht ein Projektunterverzeichnis öffnen, erzeugt Together aus dessen Inhalt automatisch ein Namespace-Diagramm mit den Namespaces, Klassen und Interfaces sowie deren Beziehungen.

So öffnen Sie ein vorhandenes Implementierungsprojekt für die Modellierung:

1. Stellen Sie sicher, dass die Together-Unterstützung aktiviert ist.
2. Wählen Sie im Hauptmenü **Datei▶Projekt öffnen**.
3. Geben Sie im Dialogfeld Projekt öffnen das Projektverzeichnis an.
4. Wählen Sie die Projekt- oder Projektgruppdatei aus .
5. Klicken Sie auf OK.

Ergebnis: Wenn die Together-Unterstützung aktiviert ist und Sie ein Implementierungsprojekt öffnen, wird der Quelltext automatisch analysiert. Anschließend werden die entsprechenden Klassendiagramme erzeugt.

Siehe auch

Together-Unterstützung aktivieren (↗ siehe Seite 248)

Projekte erstellen (↗ siehe Seite 249)

1.225 Gemeinsame Nutzung eines Projekts in TCC/TAR und RAD Studio

In diesem Abschnitt wird anhand eines C#-Projekts beschrieben, wie Modellinformationen in RAD Studio und in CodeGear Together ControlCenter (TCC) oder CodeGear Together Architect (TAR) gemeinsam verwendet werden können. Dazu erstellen Sie eine Gruppe von Diagrammen in Together ControlCenter und verweisen dann in RAD Studio auf diese Diagramme.

Zur Erstellung eines Projekts für die gemeinsame Verwendung gehen Sie folgendermaßen vor:

1. Erstellen eines C#-Projekts in RAD Studio
- 2.
3. Erstellen der Ordnerhierarchie.
4. Erstellen eines Projektes in Together ControlCenter oder Together Architect.
5. Konfigurieren von Together ControlCenter für die automatische Namespace-Zuweisung für Klassifizierer.
6. Füllen der Analyse- und Anforderungsmodelle.
7. Zugreifen auf die mit Together ControlCenter oder Together Architect erstellten Diagramme.

So erstellen Sie ein C#-Projekt:

1. Wählen Sie im Hauptmenü **Datei > Neu > Weitere**. Das Dialogfeld Objektgalerie wird geöffnet.
2. Wählen Sie als Projekttyp **C#-Projekte**
3. Wählen Sie im Bereich **Templates** die gewünschte Template aus.
4. Klicken Sie auf **OK**.
5. Geben Sie **ProjectRoot** als Projektnamen ein.
6. Geben Sie einen Speicherort für das Projekt an. Der tatsächliche Speicherort ist nicht von Bedeutung, aber Sie erstellen später dort auch das Together ControlCenter-Projekt.
7. Klicken Sie auf **OK**.

Ergebnis: Das Projekt wird erstellt und in der Projektverwaltung angezeigt.

Warnung:

So erstellen Sie die Ordnerhierarchie:

1. Navigieren Sie in Windows Explorer oder einem anderen Dateiverwaltungsfenster zum Projektordner, und erstellen Sie einen neuen Unterordner. mit der rechten Maustaste auf den Knoten **ProjectRoot**, und wählen Sie **Hinzufügen > Neuer Ordner**.
2. Geben Sie als Ordnernamen **src** ein.
3. Erstellen Sie unter dem Ordner **src** einen weiteren Ordner.
4. Geben Sie als Ordnernamen **analysis** ein.
5. Wiederholen Sie die letzten Schritte, und fügen Sie einen weiteren Ordner mit dem Namen **requirements** hinzu.
6. Speichern Sie alle Änderungen, und schließen Sie das RAD Studio-Projekt. Sie werden nun zum Speichern des Projekts aufgefordert. Klicken Sie auf **Ja**.

Die in RAD Studio erstellten Ordner werden von TCC oder TAR verwendet. Die Hierarchie beginnt mit dem Ordner **src**.

So erstellen Sie ein Projekt in Together ControlCenter oder Together Architect:

1. Starten Sie TCC bzw. TAR.
2. Sie müssen noch ein TCC- oder TAR-Projekt erstellen, damit die Diagrammdateien gemeinsam verwendet werden können. Klicken Sie im Hauptmenü auf **Datei**, und starten Sie den Experten für neue Projekte.
3. Legen Sie im ersten Fenster des Experten folgende Einstellungen fest:
 - Geben Sie den Projektnamen ein. Geben Sie für dieses Beispiel den Namen **TCC_Project** ein.
 - Geben Sie als Speicherort das Verzeichnis des RAD Studio-Projekts **ProjectRoot** ein. Wenn das RAD Studio-Projekt **ProjectRoot** beispielsweise im Verzeichnis **C:\Dokumente und Einstellungen\Benutzer\Eigene Dateien\CodeGear Studio Projects\ProjectRootC:\Programme\Microsoft Visual Studio .NET 2003\VC#\MyCSharpProjects\ProjectRoot** erstellt wurde, geben Sie diesen Pfad an. Das Verzeichnis für das TCC- bzw. TAR-Projekt muss mit demjenigen für das RAD Studio-Projekt identisch sein.
 - Wählen Sie **C#** als Standardsprache.
 - Wählen Sie **Neues Projekt** für das Projekt **Creation Scenario**.
4. Klicken Sie auf **Weiter**, um fortzufahren.
5. Geben Sie einen Pfad für die C#-Quelltextdateien an. Wählen Sie den Ordner **src** (z.B. **C:\Dokumente und Einstellungen\Benutzer\Eigene Dateien\CodeGear Studio Projects\ProjectRoot\srcC:\Programme\Microsoft Visual Studio .NET 2003\VC#\MyCSharpProjects\ProjectRoot\src**). Klicken Sie anschließend auf **Weiter**.
6. Klicken Sie auf **Nein**, um die Diagrammdateien separat zu speichern. Im Gegensatz zu Together ControlCenter werden in RAD Studio die Diagramm- und Quelltextdateien getrennt verwaltet.
7. Klicken Sie auf **Weiter**, um fortzufahren.
8. Wählen Sie im Stammverzeichnis des Projekts den Ordner **ModelSupport_%PROJEKTNNAME%ModelSupport** zum Speichern der Diagrammdateien aus (z.B. **C:\Dokumente und Einstellungen\Benutzer\Eigene Dateien\CodeGear Studio Projects\ProjectRoot\ModelSupport_ProjectRootC:\Programme\Microsoft Visual Studio .NET 2003\VC#\MyCSharpProjects\ProjectRoot\ModelSupport**).
9. Klicken Sie auf **Fertig stellen**.

Ergebnis: Die Projektstruktur wird auf der Registerkarte **Modell** des Explorers angezeigt, und die beiden Projektverzeichnisse **analysis** und **requirements** werden im Designer-Bereich angezeigt.

So konfigurieren Sie Together ControlCenter für die automatische Namespace-Zuweisung bei Klassifizierern:

1. Wählen Sie **Tools>Optionen>Standardebene** oder **Tools>Optionen>Projektebene**.
2. Erweitern Sie im linken Bereich des Dialogfelds Optionen den Knoten **Quelltext>(Sprache)**.
3. Aktivieren Sie die Option für die Namespace-Zuweisung nach Paketnamen.

Anmerkung: Damit Modelle in TCC (TAR) und RAD Studio gemeinsam genutzt werden können, unterstützt TCC (TAR) die Paket-Namespace-Organisation für C#-Projekte. Sie können bei Bedarf im Dialogfeld Optionen von TCC (TAR) das automatische Zuweisen der Namespaces für C#-Klassifizierer festlegen. Wenn Sie eine Klasse einem Paket hinzufügen (unabhängig davon, wo sich die Klasse vorher befunden hat), wird ihre neue Instanz in die Namespace-Deklaration eingefügt, die dem Paketnamen entspricht. Der Name des Namespace ist mit dem des Pakets identisch, und den in der Verzeichnis-/Paketstruktur platzierten Dateien werden die betreffenden Namespaces im Quelltext zugewiesen.

4. Klicken Sie auf **OK**, um die Einstellungen zu übernehmen und das Dialogfeld zu schließen.

Warnung: Sie können keine Pakete verschieben, da sonst die Namespace-Organisation verletzt wird. Wenn Sie ein Paket verlagern möchten, müssen Sie ein neues Paket mit demselben Namen im betreffenden Ordner erstellen, danach im neuen Paket die gewünschten Klassen erstellen und zuletzt das alte Paket entfernen.

So fügen Sie dem Modell "analysis" Elemente hinzu:

1. Doppelklicken Sie in Together ControlCenter oder Together Architect im Designer-Bereich auf das Paket **analysis**, um das Diagramm **analysis** zu öffnen.
2. Klicken Sie in der Symbolleiste auf der linken Seite des Designer-Bereichs auf die Schaltfläche **Class**.
3. Klicken Sie in den Designer-Bereich, um die Klasse dem Diagramm hinzuzufügen. Übernehmen Sie den Standardnamen **Class1**.
4. Wiederholen Sie Schritt 3, um dem Diagramm eine weitere Klasse hinzuzufügen. Übernehmen Sie den Standardnamen **Class2**.

So fügen Sie dem Modell "requirements" Elemente hinzu:

1. Doppelklicken Sie in Together ControlCenter oder Together Architect auf der Registerkarte **Modell** des Explorer-Bereichs auf das Diagramm **requirements**, um es im Designer-Bereich zu öffnen.
2. Klicken Sie in der Symbolleiste des Designer-Bereichs auf die Schaltfläche **Neues Diagramm**. Das Dialogfeld **Neues Diagramm** wird geöffnet.
3. Wählen Sie in diesem Dialogfeld das Anwendungsfalldiagramm aus.
4. Geben Sie **PlaceOrderUseCase** in das Feld für den Diagrammnamen ein.
5. Klicken Sie auf **OK**. Das Diagramm **PlaceOrderUseCase** wird im Designer-Bereich geöffnet.
6. Klicken Sie in der Symbolleiste auf der linken Seite des Designer-Bereichs auf die Schaltfläche **Actor**.
7. Klicken Sie in den Designer-Bereich, um dem Diagramm einen Akteur hinzuzufügen. Übernehmen Sie den Standardnamen **Actor1**.
8. Klicken Sie auf die Schaltfläche **Use Case**.
9. Klicken Sie in den Designer-Bereich, um dem Diagramm einen Anwendungsfall hinzuzufügen. Übernehmen Sie den Standardnamen **UseCase1**.

Ergebnis: Die Diagramme sind jetzt vollständig und können in RAD Studio verwendet werden. Die Projektstruktur wird auf der Registerkarte **Modell** des Explorer-Bereichs angezeigt.

So greifen Sie auf die mit Together ControlCenter oder Together Architect erstellten Diagramme zu:

1. Speichern Sie das Projekt, und beenden Sie Together ControlCenter bzw. Together Architect.
2. Wechseln Sie zu RAD Studio.
3. Klicken Sie im Hauptmenü auf **Datei > Zuletzt geöffnete Projekte**, und wählen Sie **ProjectRoot** aus.
4. Fügen Sie die neuen Modellelemente mit Hilfe der Projektverwaltung hinzu.
5. Die Quelltextdateien **Class1.cs** und **Class2.cs** wurden während der Arbeit in Together ControlCenter bzw. Together Architect hinzugefügt.

Anmerkung: auf die Schaltfläche **Alle Dateien anzeigen**.

6. Erweitern Sie die Ordner **src** und **analysis** in der Projektverwaltung. Die Knoten **Class1** und **Class2** sind vorhanden. Die Modellsicht wird aktualisiert, und die Quelltextdateien **Class1** und **Class2** werden in das Diagramm **analysis** übernommen. Wenn Sie nun in RAD Studio Änderungen an den Diagrammen vornehmen, werden diese in die Diagramme in TCC bzw. TAR übernommen.

Warnung: Wenn Sie in Together über ein Klassendiagramm einem Namespace ein quelltexterzeugendes Element hinzufügen (z.B. eine Klasse oder ein Interface), fügt RAD Studio dieses Element physisch in den Projektstamm ein. Together überprüft zwar nicht, wo RAD Studio die Quelltextdateien speichert, zeigt sie im Klassendiagramm aber richtig an. Verschieben Sie in der Projektverwaltung das Element per Drag&Drop in den entsprechenden Ordner, damit es bei der Arbeit in Together ControlCenter oder Together Architect an der richtigen Position angezeigt wird.

Siehe auch

Überblick zur Interoperabilität (siehe Seite 1460)

In TCC oder TAR erstellte Projekte importieren (siehe Seite 252)

1.226 Modellansicht, Diagrammansicht und Quelltext synchronisieren

Together sorgt für eine konstante Synchronisierung der verschiedenen Aspekte eines Projekts:

- Modellhierarchie in der Modellansicht
- Grafische Darstellung des Modells in der Diagrammansicht
- Quelltext (Implementierungsprojekte)

Tip: Sie können auch die Funktion **Neu laden** in der Modellansicht oder die Funktion **Aktualisieren** in der Diagrammansicht verwenden, um das komplette Modell zu aktualisieren.

Sie haben folgende Möglichkeiten zur Navigation zwischen Modellansicht, Diagrammansicht und Quelltext:

1. Wechseln von der Modellansicht zu einem Diagramm in der Diagrammansicht
2. Wechseln von der Modellansicht zu einem Modellelement in der Diagrammansicht
3. Wechseln von der Diagrammansicht zur Modellansicht
4. Wechseln von einer Lebenslinie zum zugehörigen Klassifizierer in der Modellansicht oder in einem Klassendiagramm
5. Wechseln vom Quelltext zur Modellansicht
6. Wechseln von der Modellansicht oder der Diagrammansicht zum Quelltext (in Implementierungsprojekten)
7. Bearbeiten eines synchronisierten Elements

So gelangen Sie von der Modellansicht zu einem Diagramm in der Diagrammansicht:

1. Klicken Sie in der Modellansicht mit der rechten Maustaste auf den Diagrammknoten.
2. Wählen Sie Diagramm öffnen.

Sie können auch in der Modellansicht auf den Diagrammknoten doppelklicken.

So wechseln Sie von der Modellansicht zu einem Modellelement in der Diagrammansicht:

1. Wählen Sie ein Modellelement in der Modellansicht aus.
2. Klicken Sie mit der rechten Maustaste, und wählen Sie im Kontextmenü den Befehl Im Diagramm markieren.

Anmerkung: Wenn das Modellelement in mehreren Diagrammen enthalten ist, wählen Sie das gewünschte Diagramm aus dem Untermenü.

So gelangen Sie von der Diagrammansicht zur Modellansicht:

1. Klicken Sie in der Diagrammansicht mit der rechten Maustaste auf das ausgewählte Element oder auf den Diagrammhintergrund.
2. Wählen Sie im Kontextmenü Mit Modellansicht synchronisieren.

So wechseln Sie von einer Lebenslinie zum zugehörigen Klassifizierer in der Modellansicht oder in einem Klassendiagramm:

1. Wählen Sie eine Lebenslinie in einem UML 2.0-Sequenzdiagramm in der Diagrammansicht aus, und klicken Sie mit der rechten Maustaste.
2. Wählen Sie **Auswählen > Typ in Modellansicht**, um in der Modellansicht auf den Klassifizierer zu positionieren. Oder Wählen

Sie **Auswählen** **Typ im Diagramm**, um auf den Klassifizierer in einem Klassendiagramm in der Diagrammansicht zu positionieren.

So gelangen Sie vom Quelltext in die Modellansicht:

1. Klicken Sie mit der rechten Maustaste auf die Zeile, die das gewünschte Element enthält.
2. Wählen Sie im Kontextmenü Modellansicht synchronisieren.

Ergebnis: Das entsprechende Element wird in der Modellansicht hervorgehoben angezeigt.

So gelangen Sie von der Modellansicht oder der Diagrammansicht zum Quelltext (in Implementierungsprojekten):

1. Klicken Sie mit der rechten Maustaste auf ein Modellelement oder ein Knoten-Member.
2. Wählen Sie im Kontextmenü Zur Definition gehen.

Anmerkung: Dieser Befehl steht nur für Elemente zur Verfügung, die Quelltext generieren.

Ergebnis: Der Quelltext des betreffenden Elements wird auf der Registerkarte **Editor** geöffnet. Die zugehörige Definition wird hervorgehoben angezeigt.

Tip: Wenn Sie den Quelltext einer ganzen Klasse oder eines Interface anzeigen möchten, doppelklicken Sie auf das Elementsymbol.

So bearbeiten Sie ein synchronisiertes Element:

1. Wählen Sie ein Element in der Diagrammansicht oder Modellansicht aus.
2. Bearbeiten Sie die gewünschten Felder im Objektinspektor.

Anmerkung: Sie können auch den internen Editor in der Diagrammansicht oder Modellansicht aktivieren.

Warnung: Eine Bearbeitung von Modellelementen in der Strukturansicht oder in der Projektverwaltung sollte nach Möglichkeit vermieden werden.

Siehe auch

LiveSource – Überblick (siehe Seite 1451)

Fehlerbehebung an Modellen (siehe Seite 265)

1.227 Designprojekte in Quelltext umwandeln

Diese Funktion ist nur für UML 1.5- und UML 2.0-Designprojekte verfügbar.

So erzeugen Sie Quelltext aus einem Designprojekt:

1. Wählen Sie in der Modellansicht ein Designprojekt aus.
2. Klicken Sie mit der rechten Maustaste, und wählen Sie In Quelltext umwandeln.
3. Wählen Sie im Dialogfeld Zielprojekt auswählen das gewünschte Implementierungsprojekt aus.
4. Aktivieren Sie bei Bedarf das Kontrollkästchen Namenszuordnungsdateien für Quelltexterzeugung verwenden.
5. Klicken Sie auf Umwandeln.

Ergebnis: Der Implementierungsquelltext der Klassendiagramme des Designprojekts wird dem sprachspezifischen Zielprojekt hinzugefügt. Die Diagramme werden ebenfalls in das Zielprojekt eingefügt. Auch die Diagrammstämme bleiben erhalten.

So fügen Sie Quelltext in ein Implementierungsprojekt ein:

1. Wählen Sie in der Modellansicht ein Implementierungsprojekt aus.
2. Klicken Sie mit der rechten Maustaste, und wählen Sie Quelltext aus Designprojekt umwandeln.
3. Wählen Sie im Dialogfeld Quellprojekt auswählen das gewünschte Designprojekt aus.
4. Aktivieren Sie bei Bedarf das Kontrollkästchen Namenszuordnungsdateien für Quelltexterzeugung verwenden.
5. Klicken Sie auf Umwandeln.

Ergebnis: Der Implementierungsquelltext der Klassendiagramme des Designprojekts wird dem Ziel-Implementierungsprojekt hinzugefügt. Die Diagramme werden ebenfalls in das Zielprojekt eingefügt. Auch die Diagrammstämme bleiben erhalten.

Siehe auch

Überblick zur Umwandlung in Quelltext (siehe Seite 1452)

Quellprojekt auswählen/Zielprojekt auswählen (Dialogfeld) (siehe Seite 685)

1.228 Fehlerbehebung an Modellen

Sie können ein Projekt neu aus dem Quelltext laden.

Folgende Verfahren können zur Behebung von Fehlern im Modell angewendet werden:

1. Modell aktualisieren
2. Modell neu laden
3. Modell korrigieren

So aktualisieren Sie ein Modell:

1. Öffnen Sie die Diagrammansicht.
2. Drücken Sie F6.

So laden Sie ein Modell neu:

1. Öffnen Sie die Modellansicht.
2. Klicken Sie mit der rechten Maustaste auf den Stammknoten des Projekts, und wählen Sie Neu laden.

Anmerkung: Verwenden Sie den Befehl Neu laden, wenn bei der Arbeit in Together Probleme auftreten (z.B. Elemente im Diagramm nicht mehr bearbeitet werden können oder Fehlermeldungen wie <undefinierter Wert> angezeigt werden).

Tip: Normalerweise werden bei diesen Problemen die Elemente auch nicht mehr in der Strukturansicht von RAD Studio angezeigt, und der betreffende Quelltext wird im RAD Studio-Editor blau unterstrichen. Together kann diese *verloren gegangenen* Elemente nicht immer richtig behandeln. Um die Elemente wiederherzustellen, müssen Sie den Quelltext entsprechend den im RAD Studio-Editor angezeigten Empfehlungen bearbeiten. Aktualisieren Sie in diesen Fällen das Modell mit dem Befehl **Erneut laden**, damit die Probleme nicht mehr auftreten.

So korrigieren Sie ein Modell:

1. Interaktionsdiagramme: Regenerieren Sie das Diagramm aus dem Quelltext.
2. Alle anderen Diagrammtypen: Stellen Sie sicher, dass keine erforderlichen Elemente ausgeblendet sind.

Siehe auch

Modellansicht (siehe Seite 262)

Neu laden (Befehl in Modellansicht) (siehe Seite 1270)

1.229 Mit referenzierten Projekten arbeiten

Die in binären Bibliotheken enthaltenen Klassen können in Diagrammen angezeigt werden. Sie können beispielsweise die Entitäten anzeigen, die sich in der Bibliothek `MSCorLib.dll` oder in anderen Projektreferenzen befinden. Diese Ressourcen stehen für das Projekt zu Verfügung, werden aber von Together nicht in die generierte HTML-Dokumentation für das Projekt aufgenommen.

In der Modellansicht können Klassendiagramme für die in Ihre Projekte eingebundenen Referenzen angezeigt werden. Sie können in der Projektverwaltung Referenzen zu einem Projekt hinzufügen.

So fügen Sie einem Projekt Referenzen hinzu:

1. Erweitern Sie in der Projektverwaltung den gewünschten Projektknoten.
2. Klicken Sie im Kontextmenü des Knotens **Referenzen** auf Referenz hinzufügen.

Tip: Sie können stattdessen auch im Hauptmenü **Projekt**▶**Referenz hinzufügen** wählen.

3. Wählen Sie auf der Registerkarte Projekte die zu referenzierenden Projekte aus, und klicken Sie auf Auswählen.
4. Klicken Sie anschließend auf OK.

Ergebnis: Im Dialogfeld Typ zum Instantiieren auswählen werden nun alle referenzierten Projekte angezeigt, und Sie können in ihnen die gewünschten Klassifizierer auswählen.

So zeigen Sie ein Diagramm eines referenzierten Projekts an:

1. Öffnen oder erstellen Sie ein Klassendiagramm.
2. Klicken Sie mit der rechten Maustaste auf den Diagrammhintergrund, und wählen Sie **Hinzufügen**▶**Verknüpfungen**. Das Dialogfeld Verknüpfungen bearbeiten wird geöffnet. Es enthält die für das Diagramm verfügbaren Entitäten sowie alle Entitäten außerhalb des aktuellen Namespaces.
3. Wählen Sie im linken Bereich des Dialogfelds die gewünschte Ressource in der Hierarchie aus, und klicken Sie auf Hinzufügen >>.
4. Führen Sie diesen Schritt so oft durch, bis Sie die benötigten Ressourcen hinzugefügt haben.
5. Klicken Sie auf OK, um das Dialogfeld zu schließen.

Tip: Wenn die gewünschte Ressource nicht im Dialogfeld Verknüpfungen bearbeiten angezeigt wird, wurde sie wahrscheinlich noch nicht dem Projekt als Referenz hinzugefügt. Um eine Projektreferenz hinzuzufügen, wählen Sie im Hauptmenü **Projekt**▶**Referenz hinzufügen**.

So zeigen Sie ein Diagramm für die Datei `MSCorLib.dll` an (eine Standard-DLL, die den Projekten automatisch hinzugefügt wird):

1. Erweitern Sie in der Modellansicht die Knoten Referenzen und `MSCorLib.dll`.
2. Klicken Sie mit der rechten Maustaste auf das Standarddiagramm, und wählen Sie Diagramm öffnen. Das Standarddiagramm wird in der Diagrammansicht geöffnet. Sie können die Ordner Microsoft und System erweitern, um auch die anderen Klassendiagramme anzuzeigen.

Siehe auch

Verknüpfungen erstellen (↗ siehe Seite 139)

Klassifizierer instantiiieren (↗ siehe Seite 150)

1.230 Audit-Ergebnisse exportieren

Sie können die Audit-Ergebnisse in eine XML- oder HTML-Datei exportieren, um sie den Mitgliedern Ihres Teams zu zeigen oder später zu überprüfen.

So exportieren Sie die Audit-Ergebnisse in eine Datei:

1. Markieren Sie in der Tabelle die Zeilen, die Sie speichern möchten. Wenn Sie die gesamte Tabelle exportieren möchten, nehmen Sie keine Auswahl vor.
2. Klicken Sie in der Symbolleiste auf die Schaltfläche Speichern.
3. Das Dialogfeld Audit-Ergebnis speichern wird geöffnet. Wählen Sie in der Liste Ansicht auswählen den Bereich der zu exportierenden Ergebnisse aus:
 - **Alle Ergebnisse:** Wenn die Ergebnisse gruppiert sind, wird bei Auswahl dieser Option ein Bericht für alle Gruppen auf der aktuellen Registerseite ausgegeben. Sind die Ergebnisse nicht gruppiert, werden alle Ergebnisse für die aktuelle Registerseite exportiert.
 - **Aktive Gruppe:** Wenn die Ergebnisse gruppiert sind, können Sie eine Gruppe auf der aktuellen Registerseite auswählen. Der generierte Bericht enthält dann die Ergebnisse dieser Gruppe.
 - **Ausgewählte Zeilen:** Sie können in der Ansicht mit dem Audit-Ergebnisbericht eine oder mehrere Zeilen auswählen. Bei Auswahl von **Ausgewählte Zeilen** wird ein Bericht für diese Zeilen generiert.
4. Jede Registerseite enthält eine Liste der Audits (wenn die Audits nicht gruppiert sind) bzw. eine Baumstruktur mit einer Liste für die gewählte Gruppe (bei gruppierten Audits).

Anmerkung: Die Option **Aktive Gruppe** steht nur zur Verfügung, wenn die Ergebnisse mit dem Befehl **Gruppieren nach** gruppiert wurden.

Tip: Sie können mehrere Zeilen auswählen, indem Sie mit gedrückter STRG-Taste klicken.

5. Wählen Sie in der Liste **Format auswählen** das Format für die Exportdatei.
 - **XML:** Ein XML-basierter Bericht wird generiert.
 - **HTML:** Ein HTML-basierter Bericht wird erzeugt. Bei Auswahl des HTML-Formats stehen die folgenden Kontrollkästchen zur Verfügung:
 - **Beschreibung hinzufügen:** Die Audit-Beschreibungen werden in einem eigenen Ordner gespeichert und können über Hyperlinks in der Exportdatei angezeigt werden.
 - **Browser öffnen:** Die erzeugte HTML-Datei wird im Standardanzeigeprogramm geöffnet.
6. Klicken Sie auf Speichern, um das Ergebnis in der angegebenen Exportdatei zu speichern.

Siehe auch

Überblick über Qualitätssicherungsfunktionen (siehe Seite 1456)

Audit-Ergebnisse anzeigen (siehe Seite 270)

1.231 Audit-Ergebnisse drucken

Sie können die gesamte Tabelle mit den Regelverletzungen oder nur bestimmte Zeilen und Spalten drucken.

Warnung: Diese Funktion steht nur für Implementierungsprojekte zur Verfügung.

So drucken Sie die Liste mit den Regelverletzungen:

1. Markieren Sie in der Tabelle die Zeilen, die Sie drucken möchten. Wenn Sie die gesamte Tabelle drucken möchten, nehmen Sie keine Auswahl vor.

Tip: Sie können mehrere Zeilen auswählen, indem Sie mit gedrückter STRG-Taste klicken.

2. Klicken Sie in der Symbolleiste auf die Schaltfläche **Drucken**. Das Dialogfeld **Audit drucken** wird geöffnet.

3. Wählen Sie in der Liste **Ansicht auswählen** den Bereich der zu druckenden Ergebnisse aus.

- **Alle Ergebnisse:** Wenn die Ergebnisse gruppiert sind, wird bei Auswahl dieser Option ein Bericht für alle Gruppen auf der aktuellen Registerseite ausgegeben. Sind die Ergebnisse nicht gruppiert, werden alle Ergebnisse für die aktuelle Registerseite gedruckt.
- **Aktive Gruppe:** Wenn die Ergebnisse gruppiert sind, können Sie eine Gruppe auf der aktuellen Registerseite auswählen. Der generierte Bericht enthält dann die Ergebnisse dieser Gruppe.
- **Ausgewählte Zeilen:** Sie können in der Ansicht mit dem Audit-Ergebnisbericht eine oder mehrere Zeilen auswählen. Bei Auswahl von **Ausgewählte Zeilen** wird ein Bericht für diese Zeilen generiert.

4. Jede Registerseite enthält eine Liste der Audits (wenn die Audits nicht gruppiert sind) bzw. eine Baumstruktur mit einer Liste für die gewählte Gruppe (bei gruppierten Audits).

Anmerkung: Die Option **Aktive Gruppe** steht nur zur Verfügung, wenn die Ergebnisse mit dem Befehl **Gruppieren nach** gruppiert wurden.

5. Sie können bei Bedarf im Feld **Druck-Zoom** den Zoom-Faktor für das Drucken eingeben oder die Option **An Seitengröße anpassen** aktivieren, um die Ergebnisse auf einer einzelnen Seite zu drucken. Ist **An Seitengröße anpassen** aktiviert, steht das Feld **Druck-Zoom** nicht zur Verfügung.

6. Passen Sie ggf. die Seiten- und Druckereinstellungen an:

- Klicken Sie auf das Listenfeld **Drucken**, und wählen Sie einen Drucker aus.
- Wählen Sie **Tools>Optionen**, und öffnen Sie **Together>(Ebene)>Diagramm>Drucken**. Sie können dann das Papierformat, die Ausrichtung und die Ränder festlegen.

Tip: Klicken Sie auf den Abwärtspfeil rechts neben der Option **Vorschau**, um die Druckvorschau anzuzeigen. Stellen Sie den Vorschau-Zoom mit Hilfe des Schiebereglers **Vorschau-Zoom (autom.)** ein, oder aktivieren Sie das Kontrollkästchen **Autom. Zoom der Vorschau**. Klicken Sie auf den Aufwärtspfeil rechts neben der Option **Vorschau**, um das Vorschaufenster zu schließen.

7. Klicken Sie auf **Drucken**, um das Druckdialogfeld des Betriebssystems zu öffnen und den Druckvorgang zu starten.

Siehe auch

Überblick über Qualitätssicherungsfunktionen (siehe Seite 1456)

Audit-Ergebnisse anzeigen (siehe Seite 270)

Audit drucken (Dialogfeld) (siehe Seite 698)

1.232 Audits ausführen

Audits vergleichen automatisch, ob der Quelltext den Standards oder Ihren eigenen Richtlinien bezüglich Programmierstil, Programmwartung und Stabilität entspricht. Vergewissern Sie sich vor der Ausführung von Audits, ob der Quelltext fehlerfrei compiliert wird. Wenn der Quelltext Fehler enthält oder auf Bibliotheken und Pfade nicht zugegriffen werden kann, führen die Audits nicht zum gewünschten Ergebnis.

Warnung: Diese Funktion steht nur für Implementierungsprojekte zur Verfügung.

So führen Sie Audits aus:

1. Öffnen Sie ein Implementierungsprojekt.
2. Öffnen Sie die Modellansicht.
3. Klicken Sie mit der rechten Maustaste auf den Stammknoten des Projekts. Wählen Sie im Kontextmenü QS-Audits. Das Dialogfeld Audits wird geöffnet.
4. Führen Sie in diesem Dialogfeld folgende Schritte aus:
 - Wählen Sie im Listenfeld **Bereich** den Quelltext aus, der mit dem Audit überprüft werden soll.
 - Wenn Sie **Modell** auswählen, wird das gesamte Projekt analysiert.
 - Mit **Auswahl** werden nur die Klassen, Namespaces und Diagramme berücksichtigt, die aktuell in der Diagramm- oder Modellansicht ausgewählt sind.

Tip: Falls Sie in der Diagramm- oder Modellansicht keine Elemente ausgewählt haben, wird automatisch das gesamte Projekt als **Bereich** gewählt.

5. Sollen Audits nur für bestimmte Klassen, Namespaces oder Diagramme ausgeführt werden, müssen Sie diese bereits vor dem Öffnen des Dialogfelds **Audits** auswählen.
6. Wählen Sie die auszuführenden Audits aus. Sobald Sie auf ein Audit klicken, wird dessen Beschreibung im unteren Fenster des Dialogfelds angezeigt.
7. Für jedes Audit werden der Schweregrad und die anderen audit-spezifischen Optionen auf der rechten Seite des Dialogfelds angezeigt. Ändern Sie die Einstellungen bei Bedarf.
8. Wenn Sie die gewünschten Audits ausgewählt haben, klicken Sie auf **Start**. Das Dialogfeld Operation wird ausgeführt wird angezeigt. Die Statusleiste informiert Sie über den Fortgang der Operation. Sie bleibt eingeblendet, bis der Vorgang abgeschlossen ist.
9. Mit der Schaltfläche **Abbrechen** kann der Vorgang jederzeit abgebrochen werden.

Anmerkung: Audits werden im Befehls-Thread ausgeführt. Sie können daher während der Verarbeitung nicht an Ihrem Projekt arbeiten.

Der Bericht mit den Audit-Ergebnissen wird automatisch in einem eigenen Fenster angezeigt. Sie können in der Ergebnistabelle mit der rechten Maustaste auf die Zeilen klicken und über das Kontextmenü verschiedene Operationen mit dem Bericht durchführen.

Siehe auch

Überblick über Qualitätssicherungsfunktionen (siehe Seite 1456)

Audit-Ergebnisse anzeigen (siehe Seite 270)

1.233 Audit-Ergebnisse anzeigen

Sie können Audit-Ergebnisse anzeigen und die Einträge in der Tabelle mit dem Ergebnisbericht vergleichen oder auf eine bestimmte Weise organisieren.

Die Zeilen der Tabelle sind eng mit den Diagrammelementen und dem Quelltext verknüpft. Sie können von der Tabelle aus direkt zu der Stelle im Quelltext wechseln, an der die Regelverletzung stattgefunden hat.

Warnung: Diese Funktion steht nur für Implementierungsprojekte zur Verfügung.

Nach der Anzeige der Audit-Ergebnisse haben Sie folgende Möglichkeiten:

1. Sortieren der Einträge nach den Werten einer Spalte
2. Gruppieren der Tabelleneinträge nach der aktuellen Spalte
3. Navigieren zu der Stelle mit der Regelverletzung

So sortieren Sie die Einträge nach den Werten einer Spalte:

1. Aktivieren Sie die Tabelle mit den Audit-Ergebnissen.
2. Klicken Sie auf die Spaltenüberschrift. An dem Pfeil in der Überschrift ist zu erkennen, ob die Sortierung auf- oder absteigend erfolgt.

So gruppieren Sie Tabelleneinträge nach der aktuellen Spalte:

1. Klicken Sie mit der rechten Maustaste auf die Tabelle mit den Audit-Ergebnissen, und wählen Sie **Gruppieren nach**. Sie können nun Einträge durch Ändern der Beziehung zwischen den Zeilen und Spalten organisieren.
2. Um die Gruppierung aufzuheben, klicken Sie mit der rechten Maustaste auf die Tabelle und wählen **Gruppierung aufheben**.

So navigieren Sie zu der Stelle mit der Regelverletzung:

1. Wählen Sie im Ergebnisbericht eine Regelverletzung aus.
2. Wählen Sie im Kontextmenü den Befehl **Öffnen**, oder doppelklicken Sie auf die Zeile. Sie gelangen dann direkt zur betreffenden Stelle im Quelltext.

Siehe auch

Überblick über Qualitätssicherungsfunktionen (↗ siehe Seite 1456)

Audits ausführen (↗ siehe Seite 269)

1.234 Mit Audit-Sets arbeiten

So erstellen Sie ein Audit-Set:

1. Wählen Sie im Hauptmenü den Befehl **Tools**▶**Together**▶**QS-Audits**. Das Dialogfeld QS-Audits wird geöffnet.
2. Über die Schaltflächen in der Symbolleiste des Dialogfelds können die gewünschten Operationen mit den Sets durchgeführt werden.
3. Wenn Sie ein neues Set auf Basis des Standard-Set erstellen möchten, klicken Sie auf die Schaltfläche Standard-Audit-Set festlegen.
4. Um ein bereits erstelltes Set als Ausgangspunkt zu verwenden, klicken Sie auf Set laden und wählen die gewünschte .adt-Datei aus.
5. Gehen Sie die verschiedenen Audits durch, und aktivieren Sie diejenigen, die Sie zum Set hinzufügen möchten bzw. deaktivieren Sie die Audits, die nicht benötigt werden.
6. Sie können alle Einträge in einer Gruppe auswählen, indem Sie den Gruppennamen aktivieren.
7. Wenn Sie Ihre Auswahl getroffen haben, klicken Sie auf die Schaltfläche Set speichern. Geben Sie anschließend den Pfad und den Dateinamen für die Set-Datei ein.

So führen Sie ein Audit-Set aus:

1. Wählen Sie im Hauptmenü den Befehl **Tools**▶**Together**▶**QS-Audits**. Das Dialogfeld QS-Audits wird geöffnet.
2. Klicken Sie auf die Schaltfläche Set laden, und wählen Sie die gewünschte .adt-Datei aus.
3. Klicken Sie auf Start.

Tip: Es empfiehlt sich, die .adt-Dateien immer zusammen mit den Projekten zu sichern.

Siehe auch

Überblick über Qualitätssicherungsfunktionen (☞ siehe Seite 1456)

Audit-Ergebnisse anzeigen (☞ siehe Seite 270)

1.235 Metrikdiagramme erstellen

Sie können im Bereich mit den Metrikergebnissen ein Diagramm erstellen.

Die Metrikdiagramme werden in temporären Dateien erstellt, die nach dem Schließen der Diagramme wieder gelöscht werden. Sie können jedoch die grafischen Informationen in einer Textdatei speichern, in einem bestimmten Grafikformat exportieren und als Grafiken in ein Projekt einfügen.

So erstellen Sie ein Balkendiagramm:

1. Wählen Sie die Spalte mit dem Ergebnis der gewünschten Metrik aus.
2. Klicken Sie mit der rechten Maustaste, und wählen Sie **Balkendiagramm**.

So erstellen Sie ein Kiviat-Diagramm:

1. Wählen Sie die Zeile mit dem Ergebnis des gewünschten Elements aus.
2. Klicken Sie mit der rechten Maustaste, und wählen Sie **Kiviat-Diagramm**.

So speichern Sie ein Diagramm:

1. Klicken Sie mit der rechten Maustaste auf die Registerkarte des Diagramms, und wählen Sie **Speichern**.
2. Wechseln Sie im Dialogfeld zum gewünschten Verzeichnis, und klicken Sie auf **Speichern**.

So exportieren Sie ein Diagramm in eine Bilddatei:

1. Wählen Sie das gewünschte Diagramm aus.
2. Klicken Sie im Hauptmenü auf **Datei | Diagramm als Bild exportieren**.
3. Geben Sie im Dialogfeld den Vergrößerungsfaktor und die Bildgröße an, und klicken Sie auf **Speichern**.
4. Klicken Sie auf **Speichern**.

So fügen Sie ein Diagramm einem Projekt hinzu:

1. Wählen Sie das gewünschte Diagramm aus.
2. Klicken Sie im Hauptmenü auf **Datei | <Diagrammname> in Projekt verschieben**.
3. Wählen Sie im Untermenü ein Projekt in der aktuellen Projektgruppe aus.

Siehe auch

Überblick über Qualitätssicherungsfunktionen (↗ siehe Seite 1456)

Metrikergebnisse anzeigen (↗ siehe Seite 274)

1.236 Metriken ausführen

Vergewissern Sie sich vor der Ausführung von Metriken, ob der zu analysierende Quelltext fehlerfrei compiliert wird. Wenn der Quelltext Fehler enthält oder auf Bibliotheken und Pfade nicht zugegriffen werden kann, führen die Metriken nicht zum gewünschten Ergebnis.

Warnung: Diese Funktion steht nur für Implementierungsprojekte zur Verfügung.

So führen Sie Metriken aus:

1. Öffnen Sie ein Implementierungsprojekt.
2. Öffnen Sie die Modellansicht.
3. Klicken Sie mit der rechten Maustaste auf den Stammknoten des Projekts. Wählen Sie im Kontextmenü QS-Metriken. Das Dialogfeld Metriken wird geöffnet.
4. Führen Sie in diesem Dialogfeld folgende Schritte aus:
 - Legen Sie unter **Bereich** fest, für welchen Bereich die Metriken ausgeführt werden sollen: Wenn Sie **Modell** auswählen, wird das gesamte Projekt analysiert.
 - Mit **Auswahl** werden nur die Klassen, Pakete und Diagramme berücksichtigt, die aktuell in der Diagramm- oder Modellansicht ausgewählt sind.
5. Wählen Sie die **Metriken** aus, die Sie für die Analyse verwenden möchten. Für jede Metrik wird im unteren Bereich des Dialogfelds eine Beschreibung angezeigt.

Tip: Wenn in der Diagramm- oder Navigatoransicht keine Elemente ausgewählt sind, steht der Bereich **Auswahl** nicht zur Verfügung. Um bestimmte Klassen, Pakete oder Diagramme zu überprüfen, müssen Sie diese auswählen, bevor Sie das Dialogfeld **Metriken** öffnen.

6. Im rechten Bereich des Dialogfelds **Metriken** können Sie für jede Metrik bestimmte Optionen einstellen, z.B. die Grenzwerte und die Granularität. Ändern Sie die Einstellungen bei Bedarf.
7. Wenn Sie die gewünschten Metriken ausgewählt haben, klicken Sie auf **Start**.

Anmerkung: Metriken werden im Befehls-Thread ausgeführt. Sie können daher während der Verarbeitung nicht an Ihrem Projekt arbeiten.

Ergebnis: Der Bericht mit den Metrikergebnissen wird automatisch in einem eigenen Fenster angezeigt.

Siehe auch

Überblick über Qualitätssicherungsfunktionen (↗ siehe Seite 1456)

Metrikergebnisse anzeigen (↗ siehe Seite 274)

1.237 Metrikergebnisse anzeigen

Nach der Anzeige des Metrikergebnisses haben Sie folgende Möglichkeiten:

1. Sortieren des Ergebnisses nach Spalten
2. Filtern des Metrikergebnisses
3. Aktualisieren des Ergebnisses
4. Wechseln zum Quelltext
5. Anzeigen der Metrikbeschreibung

So sortieren Sie das Metrikergebnis nach einer Spalte:

1. Wählen Sie die gewünschte Spalte in der Tabelle mit den Metrikergebnissen aus.
2. Klicken Sie auf die Spaltenüberschrift, um die Sortierreihenfolge zu ändern.

So filtern Sie das Ergebnis:

1. Um den Ergebnisbericht aussagekräftiger zu machen, können Sie die angezeigten Einträge filtern.
2. Mit den folgenden Schaltflächen in der Symbolleiste können Sie Elemente ein- und ausblenden:

Schaltfläche
Namespaces
Klassen
Methoden
Untergeordnete Elemente

So aktualisieren Sie das Ergebnis:

1. Sie können die Tabelle mit dem Metrikergebnis aktualisieren.
2. Hierfür stehen die folgenden Schaltflächen in der Tool-Palette zur Verfügung:

Schaltfläche	Beschreibung
Aktualisieren	Die angezeigten Einträge werden neu berechnet.
Neu starten	Das Dialogfeld Metriken wird geöffnet, in dem Sie neue Einstellungen festlegen und die Metrikanalyse erneut starten können.

So wechseln Sie zum Quelltext:

1. Wählen Sie im Metrikergebnis die gewünschte Zeile aus.
2. Klicken Sie mit der rechten Maustaste, und wählen Sie Öffnen. Sie gelangen dann direkt zum zugehörigen Quelltext.

So zeigen Sie die Metrikbeschreibung an:

1. Wählen Sie die Spalte mit der gewünschten Metrik in der Tabelle aus.
2. Klicken Sie mit der rechten Maustaste, und wählen Sie Beschreibung anzeigen.

Siehe auch

Überblick über Qualitätssicherungsfunktionen (siehe Seite 1456)

Metriken ausführen ( siehe Seite 273)

1.238 Mit Metrik-Sets arbeiten

So erstellen Sie ein Metrik-Set:

1. Wählen Sie im Hauptmenü den Befehl **Tools**▶**Together**▶**QS-Metriken**. Das Dialogfeld QS-Metriken wird geöffnet.
2. Die Symbolleiste des Dialogfelds enthält Schaltflächen zur Arbeit mit Metrik-Sets.
3. Wenn Sie ein neues Set auf Basis des Standard-Set erstellen möchten, klicken Sie auf die Schaltfläche Standard-Metrik-Set festlegen.
4. Um ein bereits erstelltes Set als Ausgangspunkt zu verwenden, klicken Sie auf Set laden und wählen die gewünschte .mts-Datei aus.
5. Gehen Sie die verschiedenen Metriken durch, und aktivieren Sie diejenigen, die Sie zum Set hinzufügen möchten, bzw. deaktivieren Sie die Einträge, die nicht benötigt werden.
6. Sie können alle Einträge in einer Gruppe auswählen, indem Sie den Gruppennamen aktivieren.
7. Wenn Sie Ihre Auswahl getroffen haben, klicken Sie auf die Schaltfläche Set speichern. Geben Sie anschließend den Pfad und den Dateinamen für die Set-Datei ein.

So führen Sie ein gespeichertes Set von Metriken aus:

1. Wählen Sie im Hauptmenü den Befehl **Tools**▶**Together**▶**QS-Metriken**. Das Dialogfeld QS-Metriken wird geöffnet.
2. Klicken Sie auf die Schaltfläche Set laden, und wählen Sie die gewünschte .mts-Datei aus.
3. Klicken Sie auf Start.

Tip: Es empfiehlt sich, die .mts-Dateien immer zusammen mit den Projekten zu sichern.

Siehe auch

Überblick über Qualitätssicherungsfunktionen (☞ siehe Seite 1456)

Metriken ausführen (☞ siehe Seite 273)

2 RAD Studio Dialog-Hilfe

Dieser Abschnitt enthält Hilfeinformationen für die Dialogfelder der Benutzeroberfläche von RAD Studio.

2.1 Code-Visualisierungsdiagramm

Das Code-Visualisierungsdiagramm zeigt Ihr Projekt als ein statisches UML-Strukturdiagramm an.

Die folgende Tabelle zeigt die im Diagramm verwendete UML-Standardnotation.

UML-Notation	Bedeutung
	Ein UML-Package
	Eine Klasse
	Ein Interface
+	Ein Member mit der Sichtbarkeit public
#	Ein Member mit der Sichtbarkeit protected
-	Ein Member mit der Sichtbarkeit private
—→	Eine Generalisierungsbeziehung
—→—→	Eine Realisierungsbeziehung
—→—→—→	Eine Abhängigkeitsbeziehung
—→—→—→—→	Eine Assoziationsbeziehung

Tip: Wenn das Diagramm zu groß ist, um ganz angezeigt zu werden, können Sie mit dem Fenster Übersicht zu dem gewünschten Ausschnitt navigieren.

Siehe auch

[Verwenden der Code-Visualisierung](#)

[Verwenden des Modellansichtfensters und des Code-Visualisierungsdiagramms](#)

[Verwenden des Übersichtsfensters](#)

2.2 Diagramm als Bild exportieren

Als Bild exportieren...

Verwenden Sie dieses Dialogfeld zum Erstellen einer Datei mit dem Code-Visualisierungsdiagramm. Es werden verschiedene Grafikformate (BMP, JPG und GIF) unterstützt.

Element	Beschreibung
Z	Der Zoom-Faktor (Vergrößerungsfaktor) für das Bild. Sie können in dieses Feld einen numerischen Wert eingeben oder den Schieberegler Vorschau-Zoom im Vorschaubereich verwenden. Mit dem Wert 1.0 wird ein Bild in der Originalgröße und mit dem Wert 0.5 in halber Größe erstellt. Der Wert 1.5 vergrößert das Bild 1,5 mal.
B	Die Breite des Bildes. Das Bild wird auf die neue Größe skaliert, und das Feld Zoom-Faktor wird entsprechend dem neuen Skalierungswert angepasst.
H	Die Höhe des Bildes. Das Bild wird auf die neue Größe skaliert, und das Feld Zoom-Faktor wird entsprechend dem neuen Skalierungswert angepasst.
Vorschau	Klicken Sie hier, um die Vorschau des Bildes ein- oder auszublenden. Die Vorschau zeigt an, wie das Bild mit den aktuellen Werten für Zoom-Faktor, Breite und Höhe aussieht.
Vorschau-Zoom	Mit dem Schieberegler skalieren Sie das Bild auf eine kleinere Größe. Wenn sich der Schieberegler in der äußerst linken Position befindet, wird das Bild in Vollgröße (1:1) erstellt. In der äußerst rechten Position wird das Bild 16 mal verkleinert (1:16).
Autom. Zoom der Vorschau	Markieren Sie dieses Kontrollkästchen, um beim Verschieben des Schiebereglers Vorschau-Zoom den Vorschaubereich in Echtzeit zu aktualisieren.
Speichern	Klicken Sie auf diese Schaltfläche, um das Dialogfeld Diagramm als Grafik speichern zu öffnen, in dem Sie das Grafikdateiformat für die Bilddatei auswählen können.

Siehe auch

[Verwenden der Code-Visualisierung](#)

[Verwenden des Modellansichtfensters und des Code-Visualisierungsdiagramms](#)

[Verwenden des Übersichtsfensters](#)

2.3 Komponenten-Template erzeugen

Komponenten ▶ Komponenten-Template erzeugen

Verwenden Sie dieses Dialogfeld zum Speichern der auf dem aktuellen Formular markierten Komponenten als wiederverwendbare Komponenten-Template.

Element	Beschreibung
Komponentenname	Gibt den Namen der neuen Komponenten-Template an. Per Voreinstellung wird der Name der ersten markierten Komponente angezeigt und das Wort 'Template' angehängt. Sie können diesen Namen ändern, doch achten Sie darauf, keine bereits existierenden Komponentennamen zu verwenden.
Palettenseite	Wählen Sie die Kategorie der Tool-Palette, in der die neue Template angezeigt werden soll.
PalettenSymbol	Legt das Symbol fest, das die Komponenten-Template in der Tool-Palette repräsentieren soll. Per Voreinstellung wird das Symbol der ersten markierten Komponente angezeigt. Um dies zu ändern, klicken Sie auf die Schaltfläche Ändern und wählen eine neue Datei aus. Das Bitmap darf maximal 24 mal 24 Pixel groß sein.

Siehe auch

Komponenten-Templates erzeugen (siehe Seite 60)

2.4 Experte "VCL-Komponente importieren"

Komponenten ▶ VCL-Komponente importieren

Verwenden Sie diesen Experten, um eine Typbibliothek, ein ActiveX-Element oder eine .NET-Assemblierung zu importieren.

Element	Beschreibung
Typbibliothek importieren	Klicken Sie dieses Optionsfeld an, um eine Typbibliothek zu importieren.
ActiveX-Element importieren	Klicken Sie dieses Optionsfeld an, um ein ActiveX-Steuerelement zu importieren.
.NET-Assemblierung importieren	Klicken Sie dieses Optionsfeld an, um eine .NET-Assemblierung zu importieren

Registrierte Typbibliotheken/ActiveX-Elemente/.NET-Assemblierungen

Abhängig vom Typ der Komponente, die Sie in der vorherigen Seite ausgewählt haben, werden in diesem Dialogfeld die im Global Assembly Cache registrierten Typbibliotheken, ActiveX-Elemente oder .NET-Assemblierungen angezeigt.

Element	Beschreibung
Komponentenliste	Zeigt eine Liste der im Global Assembly Cache registrierten Typbibliotheken, ActiveX-Elemente oder .NET-Assemblierungen an.
Hinzufügen	Klicken Sie diese Schaltfläche an, um einen neuen Komponente zum Hinzufügen zu suchen.

Komponente

Bearbeiten Sie auf der Seite Komponente die Parameter, die zum Importieren der Komponente in die IDE verwendet werden.

Element	Beschreibung
Klassenname	Zeigt den Namen der Klasse (oder Klassen) an, die erzeugt wird.
Palettenseite	Wählen Sie die Kategorie der Tool-Palette, in der die Komponenten erscheinen sollen.
Unit-Name	Geben Sie den Namen der Unit für die Komponente ein.
Durchsuchen	Klicken Sie diese Schaltfläche an, um den Ordner auszuwählen, in dem die Unit gespeichert werden soll.
Suchpfad	Geben Sie den Suchpfad für die neue Komponente ein. Der Vorgabewert ist der Bibliothekssuchpfad aus den Umgebungsoptionen. Der Ordner, den Sie im Feld Unit-Name angegeben haben, wird an den Suchpfad angehängt.

Installieren

Wählen Sie auf der Seite Installieren eine Aktion aus, die die IDE beim Installieren der Komponente ausführen soll.

Element	Beschreibung
Unit erstellen	Klicken Sie dieses Optionsfeld an, wenn die IDE eine neue Unit anlegen soll.
In vorhandenes Package installieren	Klicken Sie dieses Optionsfeld an, um die neue Komponente in ein bereits vorhandenes Package zu installieren.
In neues Package installieren	Klicken Sie dieses Optionsfeld an, wenn die IDE ein neues Package erstellen soll.

Anmerkung: Wenn die Plattform für die neue Komponente Win32 und das aktive Projekt ein Win32-Package ist, wird ein weiteres Optionsfeld angezeigt. Mit diesem Optionsfeld legen Sie fest, dass die neue Komponente in dem aktiven Package installiert wird. Wenn die Plattform für die neue Komponente .NET und das aktive Projekt ein .NET-Package ist, wird ein weiteres Optionsfeld angezeigt. Mit diesem Optionsfeld legen Sie fest, dass die neue Komponente in dem aktiven Package installiert wird.

2

Vorhandenes Package

Wählen Sie in diesem Dialogfeld ein vorhandenes Package aus, in das die neue Komponente installiert werden soll.

Element	Beschreibung
Liste der installierten Packages	Führt alle aktuell auf dem System installierten Packages auf.

Neues Package

Geben Sie in diesem Dialogfeld den Namen eines neuen Package an, in das die neue Komponente installiert werden soll.

Element	Beschreibung
Dateiname	Geben Sie einen Dateinamen für das neue Package an. Klicken Sie auf die Schaltfläche Durchsuchen, um einen Ordner auszuwählen, in dem das neue Package gespeichert werden soll.
Beschreibung	Geben Sie eine Beschreibung für das neue Package an.

2.5 Installieren

[Komponenten](#) ▶ [Neue VCL-Komponente](#) ▶ [Schaltfläche Installieren](#)

Verwenden Sie dieses Dialogfeld zum Installieren der neu erzeugten Komponente in ein neues oder vorhandenes Package.

In vorhandenes Package (Registerseite)

Verwenden Sie diese Registerseite, um die neu erzeugte Komponente in ein vorhandenes Package zu installieren.

Element	Beschreibung
Dateiname	Wählen Sie in der Dropdown-Liste den Namen eines installierten Package aus, oder geben Sie den Namen eines anderen Package ein.
Beschreibung	Zeigt die Package-Beschreibung an, sofern vorhanden. Ansonsten bleibt das Feld leer und ist schreibgeschützt.

In neues Package (Registerseite)

Verwenden Sie dieses Dialogfeld zum Installieren der neu erzeugten Komponente in ein zu erstellendes Package.

Element	Beschreibung
Dateiname	Geben Sie den Namen des zu erstellenden Package ein. Sie können eine Pfadangabe zu dem Namen hinzufügen; wenn Sie die Pfadangabe weglassen, wird das Package im aktuellen Verzeichnis angelegt. Package-Namen müssen innerhalb eines Projekts eindeutig sein. Klicken Sie zum Öffnen eines Datei-/Verzeichnis-Auswahlialogfeldes Durchsuchen an.
Beschreibung	Geben Sie hier optional eine kurze Beschreibung des Package ein.

2.6 Komponente installieren

Komponenten ▶ Komponente installieren

Verwenden Sie dieses Dialogfeld zum Installieren einer Komponente in ein neues oder vorhandenes Package.

In vorhandenes Package (Registerseite)

Verwenden Sie diese Registerseite, um eine Komponente in ein vorhandenes Package zu installieren.

Element	Beschreibung
Name der Unit	Geben Sie den Namen der zu installierenden Unit-Datei ein. Wenn sich die Unit im Suchpfad befindet, ist eine vollständige Pfadangabe nicht erforderlich. Befindet sich das Unit-Verzeichnis nicht im Suchpfad, wird es an das Ende des Suchpfades angehängt.
Suchpfad	Legt den Pfad fest, in dem nach Dateien gesucht wird.
Name des Package	Wählen Sie in der Dropdown-Liste den Namen eines installierten Package aus, oder geben Sie den Namen eines anderen Package ein.
Beschreibung	Geben Sie hier optional eine kurze Beschreibung der Package-Datei ein.

In neues Package (Registerseite)

Verwenden Sie diese Registerseite, um eine Komponente in ein neues Package zu installieren.

Dokumentation	Beschreibung
Name der Unit	Geben Sie den Namen der zu installierenden Unit-Datei ein. Wenn sich die Unit im Suchpfad befindet, ist eine vollständige Pfadangabe nicht erforderlich. Befindet sich das Unit-Verzeichnis nicht im Suchpfad, wird es an das Ende des Suchpfades angehängt.
Suchpfad	Legt den Pfad fest, in dem nach Dateien gesucht wird.
Name des Package	Geben Sie den Namen des zu erstellenden Package ein. Sie können eine Pfadangabe zu dem Namen hinzufügen; wenn Sie die Pfadangabe weglassen, wird das Package im aktuellen Verzeichnis angelegt. Package-Namen müssen innerhalb eines Projekts eindeutig sein. Klicken Sie zum Öffnen eines Datei-/Verzeichnis-Auswahldialogfeldes Durchsuchen an. Wenn Sie einen Dateinamen direkt in das Dialogfeld Package-Name eingeben, wird die Erweiterung .dpk automatisch hinzugefügt. Verwenden Sie stats, um auf das Package in der requires-Klausel einer anderen Package-Datei zu verweisen, oder wenn Sie das Package in einer Anwendung einsetzen.
Beschreibung	Geben Sie hier optional eine kurze Beschreibung der Package-Datei ein.

2.7 Installierte ActiveX-Komponenten

Komponenten▶Installierte .NET-Komponenten

Verwenden Sie diese Seite zum Festlegen derjenigen ActiveX-Komponenten, die in der Tool-Palette angezeigt werden sollen.

Element	Beschreibung
<input checked="" type="checkbox"/>	Gibt an, ob die Komponente in der Tool-Palette angezeigt wird. Markierte Komponenten werden angezeigt, unmarkierte nicht.
Name	Der Name der ActiveX -Komponente. Markierte Komponenten werden in der Tool-Palette angezeigt.
Kategorie	Die funktionale Kategorie, zu der die Komponente gehört. Es muss mindestens ein Komponentenname markiert sein, damit die Kategorie in der Tool-Palette erscheint.
Pfad	Der Speicherort der ActiveX -Komponente.
Typbibliothek	Der Name der Typbibliothek, in der die Komponente enthalten ist.
Kategorie	Geben Sie die Kategorie ein, zu der Sie eine ActiveX-Komponente hinzufügen möchten.
ActiveX-Komponente auswählen	Zeigt ein Dialogfeld an, in dem Sie zu einer ActiveX-Komponente navigieren können.
Zurücksetzen	Setzt die installierten Komponenten auf ihre Originalkonfiguration zurück.

Tip: Klicken Sie zum Sortieren der Anzeige eine beliebige Spaltenüberschrift an.

2.8 Suchpfade für Assemblierung

Komponenten▶Installierte .NET-Komponenten

Verwenden Sie diese Seite zum Ändern des Suchpfades, den RAD Studio beim Suchen von Assemblierungen benutzt.

Dokumentation	Beschreibung
Suchpfade	Führt die Pfade auf, die beim Erstellen einer Komponente in der Entwicklungsumgebung durchsucht werden. Wenn eine Komponente in mehreren Assemblierungen vorhanden ist, wird die Komponente aus der ersten Assemblierung, in der sie gefunden wird, verwendet. Mit den nach oben bzw. nach unten weisenden Pfeilen rechts neben der Liste mit den Suchpfaden können Sie einen markierten Pfad in der Suchreihenfolge nach oben bzw. nach unten verlagern.
Ersetzen	Ersetzt den aktuell markierten Pfad in der Liste Suchpfade durch den in dem Feld oberhalb der Schaltfläche Ersetzen enthaltenen Pfad.
Hinzufügen	Fügt den in dem Feld oberhalb der Schaltfläche Hinzufügen enthaltenen Pfad der Liste Suchpfade hinzu. Mit der Ellipsen-Schaltfläche können Sie zu einem vorhandenen Ordner navigieren.
Löschen	Entfernt den aktuell markierten Pfad aus der Liste Suchpfade.
Reset	Setzt die Suchpfade auf ihre Originalkonfiguration zurück.

Tip: Klicken Sie zum Sortieren der Anzeige eine beliebige Spaltenüberschrift an.

2.9 Installierte .NET-Komponenten

Komponenten▶Installierte .NET-Komponenten

Verwenden Sie diese Seite zum Festlegen derjenigen .NET-Komponenten, die in der Tool-Palette angezeigt werden sollen.

Dokumentation	Beschreibung
<input checked="" type="checkbox"/>	Gibt an, ob die Komponente in der Tool-Palette angezeigt wird. Markierte Komponenten werden angezeigt, unmarkierte nicht.
Name	Der Name der Komponente.
Kategorie	Die funktionale Kategorie, zu der die Komponente gehört. Es muss mindestens ein Komponentenname markiert sein, damit die Kategorie in der Tool-Palette erscheint.
Namespace	Der Namespace, zu dem die Komponente gehört.
Name der Assemblierung	Der Name der Assemblierung (.dll), in der die Komponente enthalten ist.
Pfad für Assemblierung	Der Speicherort der Assemblierung.
Kategorie	Geben Sie die Kategorie ein, zu der Sie eine Komponente hinzufügen möchten.
Assemblierung auswählen	Zeigt ein Dialogfeld an, in dem Sie zu einer Assemblierung oder einer ausführbaren Datei navigieren können.
Reset	Setzt die installierten Komponenten auf ihre Originalkonfiguration zurück.

Tip: Klicken Sie zum Sortieren der Anzeige eine beliebige Spaltenüberschrift an.

2.10 .NET VCL-Komponenten

Komponenten▶Installierte .NET-Komponenten

Verwenden Sie diese Seite zum Festlegen derjenigen .NET VCL-Komponenten, die in der Tool-Palette angezeigt werden sollen.

Dokumentation	Beschreibung
<input checked="" type="checkbox"/>	Gibt an, ob diese Komponentenkategorie in der Tool-Palette angezeigt wird.
Name	Der Name der Komponente.
Namespace/Unit	Der Namespace und die Unit, zu der die Komponente gehört.
Hinzufügen	Zeigt ein Dialogfeld an, in dem Sie zu einer DLL mit .NET VCL-Komponenten navigieren und diese installieren können.
Entfernen	Entfernt die ausgewählte Komponentenkategorie aus der Tool-Palette.
Reset	Setzt die ursprüngliche Konfiguration der installierten Komponenten wieder her.

Tip: In der Statuszeile im unteren Bereich des Dialogfeldes wird der Pfad der markierten Komponentenkategorie angezeigt.

2.11 Packages

Komponenten▶ Packages

Hier geben Sie die Entwurfszeit-Packages an, die in der IDE installiert und die Laufzeit-Packages, die auf dem System zur Verwendung in allen Projekten installiert werden sollen.

Element	Beschreibung
Entwurfs-Packages	Führt die Entwurfszeit-Packages auf, die in der IDE und für alle Projekte verfügbar sind. Überprüfen Sie die Entwurfs-Packages, die auf der gesamten Systemebene bereit gestellt werden sollen. Um die Packages auf Systemebene zu installieren, muss kein Projekt geöffnet sein.
Hinzufügen	Installiert ein Entwurfszeit-Package. Das Package ist in allen Projekten verfügbar.
Entfernen	Löscht das ausgewählte Package. Es ist dann in keinem Projekt mehr verfügbar.
Bearbeiten	Öffnet das ausgewählte Package im Package-Editor, wenn der Quelltext oder die dcp-Datei verfügbar sind.

2.12 Neue Komponente

Verwenden Sie dieses Dialogfeld zum Erstellen der Basis-Unit für eine neue Komponente.

Element	Beschreibung
Vorfahrttyp	Wählen Sie über die Dropdown-Liste eine Basisklasse aus, oder geben Sie den entsprechenden Namen ein. Wenn Sie die Eigenschaften der Komponente in der Deklaration nicht überschreiben, werden diese vom Vorfahren vererbt. Wenn Sie eine Basisklasse eingeben, werden Standardwerte in die Eingabefelder Klassename und Dateiname der Unit eingetragen. Sie können diese Einträge übernehmen oder ändern.
Klassename	Geben Sie den Namen der zu erstellenden Klasse ein. Im Allgemeinen werden alle Klassennamen mit einem T begonnen. So könnte der Name einer neuen Schaltflächenkomponente beispielsweise TMyButton lauten.
Palettenseite	Wählen Sie in der Dropdown-Liste eine Kategorie aus, oder geben Sie den entsprechenden Namen der Kategorie ein, in der die neue Komponente in der Tool-Palette erscheinen soll.
Name der Unit	Legt den Namen der Unit für die neue Komponente fest. Sie können eine Pfadangabe zu dem Namen hinzufügen; wenn Sie die Pfadangabe weglassen, wird die Unit im aktuellen Verzeichnis angelegt. Befindet sich das Unit-Verzeichnis nicht im Suchpfad, wird es an das Ende des Suchpfades angehängt.
Suchpfad	Legt den Pfad fest, in dem nach Dateien gesucht wird.
OK	Erstellt die Komponente, installiert sie aber nicht. Um die Komponente später zu installieren, wählen Sie Komponenten>Installierte .NET-Komponenten .

2.13 Experte für neue VCL-Komponenten

Komponenten▶Neue VCL-Komponente

Verwenden Sie diesen Experten zum Erstellen einer neuen VCL.NET- oder VCL Win32-Komponente.

Element	Beschreibung
Delphi für VCL.NET	Klicken Sie dieses Optionsfeld an, um eine neue Komponente für das VCL.NET-Framework und die .NET-Plattform zu erstellen.
Delphi für VCL Win32	Klicken Sie dieses Optionsfeld an, um eine neue Komponente für das VCL Win32-Framework und die Win32-Plattform zu erstellen.

Vorfahrkomponente

Auf der Seite Vorfahrkomponente wird eine Liste der installierten Komponenten angezeigt, die als Vorfahren für die neue Komponente verwendet werden können. Dieses Dialogfeld zeigt installierte Komponenten für die auf der vorherigen Seite ausgewählte Plattform an.

Element	Beschreibung
Komponentenliste	Zeigt eine Liste der installierten Komponenten für die auf der vorherigen Seite ausgewählten Plattform an.

Komponente

Auf der Seite Komponente können Sie die Parameter bearbeiten, die zum Erstellen der neuen Komponente verwendet werden.

Element	Beschreibung
Klassename	Geben Sie einen Klassennamen für die neue Komponente ein oder übernehmen Sie den Vorgabewert.
Palettenseite	Wählen Sie die Kategorie der Tool-Palette aus, in der die neue Komponente erscheinen soll.
Unit-Name	Geben Sie den Namen der Unit für die neue Komponente ein.
Suchpfad	Geben Sie den Suchpfad für die neue Komponente ein. Der Vorgabewert ist der Bibliothekssuchpfad aus den Umgebungsoptionen.

Installieren

Wählen Sie auf der Seite Installieren eine Aktion aus, die die IDE beim Installieren der neuen Komponente ausführen soll.

Element	Beschreibung
Unit anlegen	Klicken Sie dieses Optionsfeld an, wenn die IDE eine neue Unit anlegen soll.
In vorhandenes Package installieren	Klicken Sie dieses Optionsfeld an, um die neue Komponente in ein bereits vorhandenes Package zu installieren.
In neues Package installieren	Klicken Sie dieses Optionsfeld an, wenn die IDE ein neues Package erstellen soll.

Anmerkung: Wenn die Plattform für die neue Komponente Win32 und das aktive Projekt ein Win32-Package ist, wird ein weiteres Optionsfeld angezeigt. Mit diesem Optionsfeld legen Sie fest, dass die neue Komponente in dem aktiven Package installiert wird. Wenn die Plattform für die neue Komponente .NET und das aktive Projekt ein .NET-Package ist, wird ein weiteres Optionsfeld angezeigt. Mit diesem Optionsfeld legen Sie fest, dass die neue Komponente in dem aktiven Package installiert wird.

2

Vorhandenes Package

Wählen Sie in diesem Dialogfeld ein vorhandenes Package aus, in das die neue Komponente installiert werden soll.

Element	Beschreibung
Liste der installierten Packages	Führt alle aktuell auf dem System installierten Packages auf.

Neues Package

Geben Sie in diesem Dialogfeld den Namen eines neuen Package an, in das die neue Komponente installiert werden soll.

Element	Beschreibung
Dateiname	Geben Sie einen Dateinamen für das neue Package an. Klicken Sie auf die Schaltfläche Durchsuchen, um einen Ordner auszuwählen, in dem das neue Package gespeichert werden soll.
Beschreibung	Geben Sie eine Beschreibung für das neue Package an.

2.14 Felder hinzufügen

Verwenden Sie dieses Dialogfeld, um für eine Datenmenge eine persistente Feldkomponente zu erstellen.

Beim Öffnen der Datenmenge überprüft das Produkt, dass jedes nicht berechnete persistente Feld bereits besteht oder aus den Daten in der Datenmenge erstellt werden kann und erstellt dann persistente Komponenten für die angegebenen Felder. Wenn dies nicht der Fall ist, wird eine Exception ausgelöst, die Sie darüber informiert, dass das Feld nicht zulässig ist, und das Feld wird nicht geöffnet.

Nachdem Sie eine Feldkomponente für die Datenmenge angelegt haben, erstellt das Produkt für die Spalten in der zugrunde liegenden Datenbank keine dynamischen Feldkomponenten mehr.

Tip: Um mehrere Felder zu markieren, drücken Sie **STRG** und klicken die Felder an. Um einen Felderbereich zu markieren, drücken Sie **UMSCHALT** und klicken das erste und das letzte Feld des Bereichs an.

2.15 Lokale Daten zuweisen

Verwenden Sie dieses Dialogfeld, um die aktuelle Datensatzgruppe von einer anderen Client-Datenmenge in die ausgewählte Client-Datenmenge zu kopieren. Dies ist beispielsweise beim Verwenden einer Client-Datenmenge als Nachschlagetabelle oder beim Testen der Datenmenge zur Entwurfszeit hilfreich. Wählen Sie die zu kopierende Datenmenge in der Liste der verfügbaren Datenmengen aus, und klicken Sie auf OK. Um die Datensätze einer Client-Datenmenge zur Entwurfszeit zu löschen, setzen Sie deren Eigenschaft Active auf True, klicken mit der rechten Maustaste auf die Datenmenge, und wählen Daten löschen.

2.16 Spaltensammlungs-Editor

Verwenden Sie dieses Dialogfeld, um Spalten in einem *DataTable*-Objekt zu erstellen und zu entfernen und um die Eigenschaften der Datenmengenspalten zu konfigurieren. Wenn Sie die Eigenschaft *Active* des Datenadapters auf **True** setzen, werden die Namen der Datenmengenspalten - so wie sie von der Datenbank zurückgegeben werden - in der Elementliste angezeigt.

Element	Beschreibung
Hinzufügen	Fügt dem <i>DataTable</i> -Objekt der Datenmenge eine Spalte hinzu.
Entfernen	Entfernt die ausgewählte Spalte aus dem <i>DataTable</i> -Objekt der Datenmenge.
Eigenschaften	Enthält eine Liste der Datenbankeigenschaften, die Sie für die Datenmengenspalten verwenden können. Die Eigenschaften ändern sich je nach Datenbanktyp.
Eigenschaften Spaltenzuordnung	Beim Zuordnen der Datenmengenspalten einer XML-Datei setzen Sie diese Eigenschaft, um festzulegen, ob der Spaltenwert als Element, als Attribut oder als einfacher Inhalt behandelt oder verborgen werden soll.

Siehe auch

[Standarddatenmengen verwenden](#)

2.17 Bedingungssammlungs-Editor

Verwenden Sie dieses Dialogfeld zum Hinzufügen, Konfigurieren und Entfernen von Bedingungen in den Datenmengenspalten eines *DataTable*-Objekts. In diesem Editor können Sie auch eindeutige oder Fremdschlüsselbedingungen verwalten.

Element	Beschreibung
Hinzufügen	Fügt der <i>DataTable</i> -Spalte eine eindeutige oder Fremdschlüsselbedingung hinzu.
Bearbeiten	Diese Schaltfläche erscheint, nachdem Sie eine Bedingung hinzugefügt haben. Sie können damit die in der Bedingung angegebenen Spalten ändern.
Entfernen	Entfernt die ausgewählte Bedingung aus einer <i>DataTable</i> -Spalte.
Eigenschaften	Die Eigenschaftsliste ist schreibgeschützt, weil alle hier angezeigten Eigenschaften von vorhandenen Datenmengen oder Spaltendefinitionen abgeleitet sind. Sie können aber die Auswahl in dieser Liste erweitern, um die Eigenschaften einer bestimmten Spalte und der Bedingung selbst anzuzeigen.

Siehe auch

[Standarddatenmengen verwenden](#)

2.18 Beziehungskollektions-Editor

Verwenden Sie dieses Dialogfeld zum Erstellen, Bearbeiten und Löschen von Beziehungen zwischen Tabellen in der aktuellen Datenmenge.

Element	Beschreibung
Hinzufügen	Fügt der Kollektion eine neue Beziehung hinzu. Zeigt das Dialogfeld Beziehung an, in dem Sie einen Beziehungsnamen, Schlüsselfelder und Regeln für Tabellen in der Datenmenge festlegen können.
Bearbeiten	Zeigt das Dialogfeld Beziehung an, in dem Sie eine vorhandene Tabellenbeziehung ändern können.
Entfernen	Entfernt die markierte Beziehung aus der Elementliste.
Eigenschaften	Stellt eine schreibgeschützte Liste der Eigenschaften bereit, die von der Datenmenge und den für die Beziehung verwendeten Spalten abgeleitet sind.

Siehe auch

[Standarddatenmengen verwenden](#)

2.19 Tabellensammlungs-Editor

Verwenden Sie dieses Dialogfeld, um der Datenmenge *DataTable*-Objekten hinzuzufügen oder daraus zu entfernen und um die Eigenschaften der *DataTable*-Objekte zu setzen. Nur wenn die Eigenschaft *Active* des Datenadapters auf **True** gesetzt ist, wird in der Elementliste automatisch ein Element für jedes *DataTable*-Objekt in der korrespondierenden Datenmenge angezeigt.

Element	Beschreibung
Hinzufügen	Fügt der Datenmenge ein <i>DataTable</i> -Objekt hinzu.
Entfernen	Entfernt das ausgewählte <i>DataTable</i> -Objekt aus der Datenmenge.

Siehe auch

[Standarddatenmengen verwenden](#)

2.20 Anweisungstext-Editor

Verwenden Sie dieses Dialogfeld zur Erstellung des Befehlstexts für die Eigenschaft CommandText des BdpCommand-Objekts.

Element	Beschreibung
Verbindung	Bezeichnet eine Verbindung in einer Liste mit BdpConnection-Objekten. Dieses Element muss als erstes ausgewählt werden. Die anderen Textfelder werden dann automatisch mit entsprechenden Daten gefüllt.
Auswählen	Veranlasst die Erzeugung einer SQL Select-Anweisung durch das BdpCommandBuilder-Objekt. Diese Anweisung muss vorhanden sein, damit das BdpCommandBuilder-Objekt andere SQL-Anweisungen (Update, Insert oder Delete) generieren kann. Die Select-Anweisung kann entweder direkt im BdpCommand-Objekt bereitgestellt oder mit Hilfe dieses Dialogfelds erzeugt werden.
Aktualisieren	Legt fest, dass das BdpCommandBuilder-Objekt basierend auf der Select-Anweisung eine Update-Anweisung generiert.
Einfügen	Legt fest, dass das BdpCommandBuilder-Objekt basierend auf der Select-Anweisung eine Insert-Anweisung generiert.
Löschen	Legt fest, dass das BdpCommandBuilder-Objekt basierend auf der Select-Anweisung eine Delete-Anweisung generiert.
SQL generieren	Erzeugt basierend auf den aktivierten Kontrollkästchen SQL-Anweisungen.
Optimieren	Legt fest, dass das BdpCommandBuilder-Objekt nach Möglichkeit optimierte SQL-Anweisungen für die getroffenen Einstellungen erzeugt.
Tables	Zeigt eine Liste mit den Tabellen der aktuellen Datenbank an, die vom aktuellen BdpConnection-Objekt repräsentiert wird. Wenn Sie eine Tabelle in dieser Liste markieren, werden die Spalten der Tabelle angezeigt. Wählen Sie einen Tabellenamen aus, um die Tabelle zur SQL-Anweisung im SQL-Feld hinzuzufügen.
Spalten	Zeigt eine Liste mit den Spalten der ausgewählten Tabelle an. Wählen Sie einen oder mehrere Spaltennamen aus, um Spalten zur SQL-Anweisung im SQL-Feld hinzuzufügen.

2.21 Anweisungstext-Editor

Verwenden Sie dieses Dialogfeld, um den Anweisungstext (SQL-Anweisung) für Datenmengen-Komponenten zu erstellen, die die Eigenschaft CommandText besitzen. Mithilfe der Schaltflächen des Dialogfeldes können Sie SELECT- und FROM-Klauseln hinzufügen; andere Klauseln (WHERE, GROUP BY, HAVING, ORDER BY usw.) müssen Sie manuell in das Eingabefeld SQL eingeben.

Element	Beschreibung
Tables	Zeigt die Namen der Tabellen an, die in der aktuellen Datenbank zur Verfügung stehen. Markieren Sie eine Tabelle und klicken Sie die Schaltfläche Tabelle zu SQL hinzufügen an, um die angezeigte SQL-Anweisung in das Eingabefeld SQL zu übernehmen.
Tabelle zu hinzufügen	Fügt eine SELECT-Klausel für die in der Liste Tabellen markierte Tabelle in das Eingabefeld SQL ein.
Felder	Zeigt die Namen der Spalten in der Tabelle an, die derzeit in der Liste Tabellen markiert ist. Markieren Sie eine Spalte und klicken Sie die Schaltfläche Feld zu SQL hinzufügen an, um die angezeigte SQL-Anweisung in das Eingabefeld SQL zu übernehmen. Um mehrere Spalten zu markieren, drücken Sie STRG und klicken die Spalten an. Um einen Spaltenbereich zu markieren, drücken Sie UMSCHALT und klicken die erste und die letzte Spalte des Bereichs an.
Feld zu hinzufügen	Fügt die in der Liste Felder markierten Spalten der SELECT-Klausel hinzu.
SQL	Zeigt den Namen der Anweisung (SQL-Anweisung) für die CommandText-Eigenschaft der Datenmenge oder die Anweisungskomponente an. Diese Anweisung kann manuell bearbeitet werden.

2.22 Datenadapter konfigurieren

Verwenden Sie dieses Dialogfeld zur Erstellung des Befehlstexts für die Eigenschaft DataAdapter des BdpDataAdapter-Objekts.

Element	Beschreibung
Verbindung	Bezeichnet eine Verbindung aus einer Liste von Live-BdpConnection-Objekten. Dieses Element muss als erstes ausgewählt werden. Die anderen Textfelder werden dann automatisch mit entsprechenden Daten gefüllt.
Auswählen	Veranlasst die Erzeugung einer SQL Select-Anweisung durch das BdpCommandBuilder-Objekt. Diese Anweisung muss vorhanden sein, damit das BdpCommandBuilder-Objekt andere SQL-Anweisungen (Update, Insert oder Delete) generieren kann. Die Select-Anweisung kann entweder direkt im BdpCommand-Objekt bereitgestellt oder mit Hilfe dieses Dialogfelds erzeugt werden.
Aktualisieren	Legt fest, dass das BdpCommandBuilder-Objekt basierend auf der Select-Anweisung eine Update-Anweisung generiert.
Einfügen	Legt fest, dass das BdpCommandBuilder-Objekt basierend auf der Select-Anweisung eine Insert-Anweisung generiert.
Löschen	Legt fest, dass das BdpCommandBuilder-Objekt basierend auf der Select-Anweisung eine Delete-Anweisung generiert.
SQL generieren	Erzeugt basierend auf den aktivierten Kontrollkästchen SQL-Anweisungen.
Optimieren	Legt fest, dass das BdpCommandBuilder-Objekt nach Möglichkeit optimierte SQL-Anweisungen für die getroffenen Einstellungen erzeugt.
Tables	Zeigt eine Liste mit den Tabellen der aktuellen Datenbank an, die vom aktuellen BdpConnection-Objekt repräsentiert wird. Wenn Sie eine Tabelle in dieser Liste markieren, werden die Spalten der Tabelle angezeigt. Wählen Sie einen Tabellenamen aus, um die Tabelle zur SQL-Anweisung im SQL-Feld hinzuzufügen.
Spalten	Zeigt eine Liste mit den Spalten der ausgewählten Tabelle an. Wählen Sie einen oder mehrere Spaltennamen aus, um Spalten zur SQL-Anweisung im SQL-Feld hinzuzufügen.

2.23 Verbindungseditor

Verwenden Sie dieses Dialogfeld zur Auswahl einer Verbindungskonfiguration oder zur Bearbeitung benannter Verbindungen, die in der Datei bdpConnections.xml gespeichert sind. Mit dem Editor können Verbindungen hinzugefügt, gelöscht und getestet werden.

Element	Beschreibung
Verbindungen	Enthält alle benannten Verbindungskonfigurationen für den aktuell ausgewählten Treiber. Wenn Sie eine Verbindung in dieser Liste auswählen, werden in der Tabelle "Einstellungen" die zugehörigen Verbindungsparameter angezeigt.
Einstellungen	Zeigt die Einstellungen für die ausgewählte Verbindungskonfiguration an. Die Spalte "Name" enthält die Namen der Verbindungsparameter, die für den Treiber der aktuellen Verbindung angegeben werden können. In der Spalte "Wert" ist der Wert des jeweiligen Verbindungsparameters angegeben.
Hinzufügen	Ermöglicht das Hinzufügen einer neuen benannten Verbindung. Durch Klicken auf die Schaltfläche Hinzufügen öffnen Sie das Dialogfeld Neue Verbindung, in dem Sie den Namen für die Verbindungskonfiguration und die zu verwendenden Treibern angeben können.
Entfernen	Löscht die Verbindung, die in der Liste Verbindungen ausgewählt ist. Die Verbindung wird gleichzeitig aus der Datei bdpConnections.xml entfernt und kann anschließend nicht mehr verwendet werden.
Test	Versucht, mit den aktuellen Einstellungen eine Verbindung zum Datenbankserver herzustellen. Mit Hilfe dieser Schaltfläche können Sie Ihre Verbindungskonfiguration testen.

2.24 Verbindungseditor

Verwenden Sie dieses Dialogfeld, um eine Verbindungskonfiguration für eine TSQLConnection-Komponente auszuwählen oder die in der Datei dbxconnections.ini gespeicherten benannten Verbindungen zu bearbeiten. Alle Änderungen in diesem Dialogfeld werden in die Datei dbxconnections geschrieben, sobald Sie auf OK klicken. Die markierte Verbindung wird auch als Wert der Eigenschaft ConnectionName der SQL-Verbindungskomponente übernommen.

Element	Beschreibung
Verbindung hinzufügen	Zeigt das Dialogfeld Neue Verbindung hinzufügen an, in dem Sie den Namen für die Verbindungskonfiguration und den zu verwendenden Treiber festlegen können. Die neue Verbindungskonfiguration wird der Datei dbxconnections.ini hinzugefügt.
Verbindung löschen	Löscht die Verbindung, die in der Liste Verbindungen ausgewählt ist. Die Verbindung wird gleichzeitig aus der Datei bdpConnections.xml entfernt und kann anschließend nicht mehr verwendet werden.
Verbindung umbenennen	Öffnet das Dialogfeld Verbindung umbenennen, in dem Sie einen neuen Namen für die in der Liste Verbindungen ausgewählte Konfiguration eingeben können. Da der Name auch in der Datei dbxconnections.ini geändert wird, kann der alte Name anschließend nicht mehr verwendet werden.
Verbindung testen	Versucht, mit den aktuellen Einstellungen eine Verbindung zum Datenbankserver herzustellen. Mit Hilfe dieser Schaltfläche können Sie Ihre Konfiguration testen.
Einstellungen anzeigen	Öffnet das Dialogfeld Treibereinstellungen, das Informationen über die aktuell installierten Treiber anzeigt.
Treibername	Führt alle Treiber auf, für die ein Verbindung definiert ist. Wenn Sie einen Treiber auswählen, werden in der Liste Verbindungen nur die für diesen Eintrag konfigurierten Verbindungseinstellungen angezeigt.
Einstellungen	Zeigt die Einstellungen für die ausgewählte Verbindungskonfiguration an. Die Spalte Schlüssel zeigt die Namen aller Verbindungsparameter, die für den Treiber der aktuellen Verbindung angegeben werden können. In der Spalte Wert können die Einstellungen beliebig geändert werden. Ändern Sie den Treibernamen einer Verbindung nicht. Die angezeigten Einstellungen können sonst möglicherweise nicht für den neuen Treiber verwendet werden.
Name der Verbindung	Enthält alle benannten Verbindungskonfigurationen für den aktuell ausgewählten Treiber. Sie können einen beliebigen Eintrag markieren und anschließend die Einstellungen ändern oder auf OK klicken, um die Konfiguration der Eigenschaft ConnectionName zuzuweisen.

2.25 Editor für Verbindungs-Strings (ADO)

Verwenden Sie dieses Dialogfeld, um den String für eine Verbindung einer ADO-Datenkomponente zu einem ADO-Datenspeicher festzulegen. Sie können den Verbindungs-String eingeben, mithilfe eines ADO-Dialogfelds zusammenstellen oder den String in einer Datei platzieren.

Element	Beschreibung
Datenverknüpfungsdatei verwenden	Markieren Sie den Namen der Verknüpfungsdatei oder geben Sie ihn ein, oder klicken Sie auf die Schaltfläche Durchsuchen, um die Datei zu suchen.
Verbindungs-String verwenden	Geben Sie den Verbindungs-String mit den Verbindungsinformationen in das Eingabefeld ein. Alternativ können Sie auch auf die Schaltfläche Aufbauen klicken, um das Dialogfeld Datenverknüpfungseigenschaften anzuzeigen, das Sie durch den Vorgang des Einrichtens und Testens der Verbindung leitet.

Tip: ADO unterstützt in Verbindungs-Strings die folgenden vier Argumente: Provider, Dateiname, externer Provider und externer Server. Alle anderen Argumente werden an den Provider übergeben. Dazu können die Benutzer-ID, das Anmeldekennwort, der Name der Standarddatenbank, persistente Sicherheitsinformationen, Namen von ODBC-Datenquellen, Verbindungs-Timeout-Werte und lokale Bezeichner zählen. Diese Parameter und ihre Werte sind für bestimmte Provider, Server und ODBC-Treiber spezifisch, nicht für ADO oder Delphi. Genauere Informationen über diese Parameter finden Sie in der Dokumentation des Providers, Servers oder ODBC-Treibers.

2.26 Datenadapter-Konfiguration DataSet

Verwenden Sie diese Registerseite, um dem Datenadapter eine Datenmenge zuzuordnen.

Element	Beschreibung
Neues DataSet	Legt eine neue Datenmenge fest, wenn noch keine vorhanden ist.
Vorhandenes DataSet	Legt eine vorhandene Datenmenge fest, die dem Datenadapter zugeordnet ist.
Ohne	Legt keine Datenmenge fest.

2.27 Datenadapter-Konfiguration Daten in der Vorabansicht

Verwenden Sie dieses Dialogfeld zum Anzeigen der Ergebnismenge, die von der aktuellen SQL-Select-Anweisung zurückgegeben wird. Mithilfe der Vorabansicht können Sie Ihre SQL-Anweisung verändern, um eine genauere Ergebnismenge zu erhalten, die dann in die Datenmenge verlagert oder aus der Datenmenge abgerufen werden kann. Diese Registerseite befindet sich in den Dialogfeldern Anweisungstext-Editor oder Datenadapter-Konfiguration.

Element	Beschreibung
Zeilen begrenzen	Legt eine Zeilenlimitoption für die aktuelle SQL-Anweisung fest.
Abzurufende Zeilen	Legt die maximale Zeilenanzahl fest.
Aktualisieren	Aktualisiert die in der Liste angezeigte Ergebnismenge, indem die SQL-Anweisung erneut ausgeführt wird.

2.28 Datenbank-Editor

Verwenden Sie dieses Dialogfeld zum Einrichten der Verbindung zu einer Datenbank.

Element	Beschreibung
Name	Der Name der Datenbank. Über diesen Namen wird die Datenbankkomponente im Anwendungscode angesprochen.
Aliasname	Der BDE-Alias der Datenbank. Wählen Sie einen Alias aus der Dropdown-Liste. Die Liste enthält alle Aliase, die derzeit bei der BDE registriert sind. Wenn Sie keine Verbindung zu einer als BDE-Alias registrierten Datenbank herstellen möchten, weisen Sie statt dessen der Eigenschaft Treiber einen Wert zu. Wenn Sie der Eigenschaft Alias einen Wert zuweisen, wird die Eigenschaft Driver gelöscht, da der Treibertyp im BDE-Alias integriert ist.
Treibername	Der Datenbanktyp, den die Datenbankkomponente repräsentiert. Wählen Sie in der Dropdown-Liste einen der Treibertypen STANDARD, ORACLE, SYBASE oder INTERBASE. Wenn der Datenbankserver als Alias bei der BDE registriert ist, können Sie statt dessen die Option Alias verwenden. Wenn Sie einen Treiber wählen, wird die Eigenschaft Alias gelöscht, damit keine Konflikte mit dem im Alias integrierten Treiber auftreten.
Parameter überschreibt	Die Werte aller Anmeldeparameter, die beim Herstellen der Verbindung benötigt werden. Die einzelnen Parameter hängen vom Typ der Datenbank ab. Eine Liste aller Parameter und ihrer Standardwerte erhalten Sie mit der Schaltfläche Vorgabe. In der Liste können Sie die Standardwerte ändern.
Vorgaben	Setzt die Parameter auf die Standardwerte für den Treibertyp.
Löschen	Entfernt alle überschriebenen Parameter.
Anmeldeaufforderung	Bewirkt, dass ein Anmeldedialogfeld automatisch eingeblendet wird, wenn der Benutzer eine Verbindung zu der Datenbank herstellt. Deaktivieren Sie die Option, wenn Sie den automatischen Anmeldedialog unterdrücken möchten. Die meisten Datenbankserver (mit Ausnahme der dateibasierten STANDARD-Typen) verlangen vom Benutzer bei der Anmeldung ein Kennwort. Wenn Sie bei diesen Servern auf den automatischen Anmeldedialog verzichten, muss die Anwendung den Benutzernamen und das Kennwort auf andere Weise bereitstellen. Dazu eignen sich beispielsweise hart codierte Parameter oder eine Ereignisbehandlungsroutine für OnLogin, in der den entsprechenden Parametern Werte zugewiesen werden.
Inaktive Verbindung halten	Legt fest, dass die Anwendung auch dann mit der Datenbank verbunden bleibt, wenn keine Datenmengen geöffnet sind. Bei Verbindungen zu externen Datenbankservern oder bei Anwendungen, die häufig Datenmengen öffnen und schließen, sollten Sie das Kontrollkästchen aktivieren. Sie verringern dadurch den Netzwerkverkehr, beschleunigen die Anwendung und umgehen die ständige Neuanmeldung beim Wiederherstellen der Verbindung. Wenn Sie das Kontrollkästchen deaktivieren, wird die Verbindung beendet, sobald die letzte Datenmenge geschlossen wird. Dieses Vorgehen spart Systemressourcen, hat jedoch den Nachteil, dass die Verbindung neu eingerichtet und initialisiert werden muss, wenn wieder eine Datenmenge geöffnet wird.

2.29 Datenbankformular-Experte

Datei▶**Neu**▶**Weitere**▶**Delphi-Projekte**▶**Business**▶**Datenbankformular-Experte**

Verwenden Sie diesen Experten zum Anlegen eines Formulars, das Daten aus einer lokalen oder externen Datenbank anzeigt. Der Experte verbindet das Formular mit einer *TTable*- oder *TQuery*-Komponente, schreibt SQL-Anweisungen für *TQuery*-Komponenten, definiert die Tabulatorreihenfolge für das Formular und verbindet *TDataSource*-Komponenten mit *TTable*/*TQuery*-Komponenten.

Folgen Sie den Anweisungen auf den einzelnen Registerseiten des Experten. Klicken Sie auf Weiter, um mit der nächsten Seite fortzufahren. Klicken Sie dann auf Fertig stellen, um das Formular auf der Grundlage der angegebenen Informationen zu erzeugen.

Tip: Der Datenbankformular-Experte kann auch über das Menü **Datenbank**▶**Formular-Experte** aufgerufen werden.

2.30 Eigenschaften der Datenmenge

Verwenden Sie dieses Dialogfeld, um Tabellen, Spalten und andere Eigenschaften der Datenmenge zu untersuchen.

Im linken Bereich des Dialogfeldes wird die Datenmenge mit ihren Tabellen angezeigt. Markieren Sie im linken Bereich ein Objekt, um dessen Eigenschaften in einem schreibgeschützten Gitter im rechten Bereich anzuzeigen.

2.31 Treibereinstellungen

Verwenden Sie dieses Dialogfeld, um anzuzeigen, welche Dateien zu den verschiedenen dbExpress-Treibern gehören. Diese Zuordnungen werden in der Datei dbxdrivers.ini gespeichert.

Element	Beschreibung	
Treibername	Zeigt die Namen aller Treiber in der Datei dbxdrivers.ini an. Die Namen entsprechen den möglichen Werten des Parameters DriverName im Verbindungseditor und können der Eigenschaft DriverName einer TSQLConnection-Komponente zugewiesen werden.	
Bibliotheksname	Enthält die dbExpress-Treiberdatei für den Treiber in der Spalte Treibername. Dieser Name wird automatisch der Eigenschaft LibraryName einer TSQLConnection-Komponente zugewiesen, wenn Sie den Wert für DriverName angeben.	
Bibliothek Herstellers	des	Zeigt die client-seitige DLL für den Datenbankserver an, der dem spezifischen Treiber zugeordnet ist. Die DLL wird vom Hersteller der Datenbank bereitgestellt. Dieser Eintrag wird automatisch der Eigenschaft VendorLib einer TSQLConnection-Komponente zugewiesen, wenn Sie den Wert für DriverName angeben.

2.32 Feldverbindungs-Designer

Verwenden Sie dieses Dialogfeld, um eine Haupt/Detail-Beziehung zwischen zwei Tabellen einzurichten.

Element	Beschreibung
Verfügbare Indizes	(Wird nicht für Tabellen auf einem Datenbankserver angezeigt.) Zeigt den aktuell gewählten Index zum Verbinden der Tabellen an. Wenn Sie keinen anderen Indexnamen in der Eigenschaft IndexName der Tabelle eingeben, ist der Standardindex der Verknüpfung der primäre Index für die Tabelle. Weitere verfügbare Indizes, die in der Tabelle definiert sind, können aus der Dropdown-Liste ausgewählt werden:
Detailfelder	Enthält die Detailfelder der Tabelle.
Hinzufügen	Fügt die ausgewählten Detail- und Hauptfelder dem Feld Verknüpfte Felder hinzu.
Hauptfelder	Enthält die Hauptfelder der Tabelle.
Verknüpfte Felder	Zeigt die ausgewählten Haupt- und Detailfelder an.
Löschen	Entfernt die ausgewählten Zeilen aus dem Feld Verknüpfte Felder.
Löschen	Entfernt alle Zeilen aus dem Feld Verknüpfte Felder.

2.33 Felder-Editor

Mit dem Felder-Editor können Sie einer Datenmenge neue persistente Felder hinzufügen und Datenfelder, berechnete Felder und Nachschlagefelder erstellen. Dieses Dialogfeld wird angezeigt, wenn Sie eine der Datenmengenkomponenten mit der rechten Maustaste anklicken, z.B. TADODataSet oder TSQLDataSet und Felder-Editor wählen.

Verwenden Sie den Felder-Editor während des Entwurfs, um persistente Listen von den Feldkomponenten zu erstellen, die von den Datenmengen in Ihrer Anwendung benutzt werden. Die persistenten Feldkomponentenlisten werden in Ihrer Anwendung gespeichert und ändern sich auch dann nicht, wenn die Struktur einer Datenmenge zugrundeliegenden Datenbank modifiziert wird. Alle Felder in einer Datenmenge sind entweder persistent oder dynamisch.

Kontextmenü

Rechtsklicken Sie im Felder-Editor, um die folgenden Befehle anzuzeigen.

Element	Beschreibung
Felder hinzufügen	Verwenden Sie diesen Befehl, um das Dialogfeld Felder hinzufügen anzuzeigen, mit dem Sie einer Datenmenge persistente Feldkomponenten hinzufügen können.
Neues Feld	Verwenden Sie diesen Befehl, um das Dialogfeld Neues Feld anzuzeigen, mit dem Sie neue persistente Felder hinzufügen oder bereits in der Datenmenge vorhandene ersetzen können. Folgende Typen von persistenten Feldern lassen sich erzeugen: Datenfelder, berechnete Felder und Nachschlagefelder.
Alle Felder hinzufügen	Erstellt in der zugrundeliegenden Datenmenge für jedes Feld persistente Felder.
Ausschneiden	Entfernt die ausgewählten Felder aus dem Felder-Editor und fügt diese in die Zwischenablage ein.
Einfügen	Fügt die Inhalte aus der Zwischenablage in den Felder-Editor ein.
Löschen	Löscht die ausgewählten Felder, ohne diese in die Zwischenablage zu kopieren.
Alles auswählen	Wählt alle Felder im Felder-Editor aus.

2.34 Fremdschlüsselbedingung

Verwenden Sie dieses Dialogfeld zum Definieren einer Fremdschlüsselbedingung zwischen Tabellen der Datenmenge. Die zum Zeitpunkt des Öffnens dieses Dialogfeldes ausgewählte Tabelle wird automatisch auf die Fremdschlüsseltabelle gesetzt. Sie könnten daher versuchen, die Haupttabelle in einer Haupt/Detail-Beziehung als Detailtabelle festzulegen. Das würde zumindest unvorhersehbare Ergebnisse zurückliefern. Wenn die Fremdschlüsselspalte der eigentlichen Detail- oder Fremdschlüsseltabelle Duplikate enthält, könnten Entwurfszeitfehler generiert werden.

Element	Beschreibung
Name	Legen Sie den Namen der Beziehung fest.
Übergeordnete Tabelle	Geben Sie die in der Beziehung übergeordnete Tabelle an.
Untergeordnete Tabelle	Legt die in der Beziehung untergeordnete Tabelle fest. Dieser Wert wird von der Tabelle, die beim Öffnen dieses Dialogfeldes ausgewählt war, bestimmt. Der Wert ist schreibgeschützt.
Schlüsselspalten	Wählen Sie eine oder mehrere Spalten aus, die als Primärschlüssel in der übergeordneten Tabelle dienen sollen.
Fremdschlüsselspalten	Wählen Sie eine oder mehrere Spalten aus, die als Fremdschlüssel in der untergeordneten Tabelle dienen sollen. Diese Spalten müssen entweder bezüglich des Namens oder des Datentyps und Werts mit denjenigen korrespondieren, die Sie als Primärschlüssel ausgewählt haben.
Aktualisierungsregel	Wählen Sie die Regel zum Aktualisieren von Datensätzen aus. Bezieht sich auf Haupt/Detail-Beziehungen und darauf, wie Detaildatensätze aktualisiert werden, wenn ein Hauptdatensatz aktualisiert wurde.
Löschregel	Wählen Sie die Regel zum Löschen von Datensätzen aus. Bezieht sich auf Haupt/Detail-Beziehungen und darauf, wie Detaildatensätze gelöscht werden, wenn ein Hauptdatensatz gelöscht wurde.
Annehmen/Verwerfen-Regel	Wählen Sie die Annehmen/Verwerfen-Regel aus, die beim Einfügen in eine Haupt/Detail-Beziehung verwendet werden soll.

Siehe auch

[Standarddatenmengen verwenden](#)

2.35 Datenmenge erzeugen

Verwenden Sie dieses Dialogfeld, um Datensätze in der Datenmenge zu sortieren, Datensätze schnell zu suchen, sichtbare Datensätze festzulegen und Haupt/Detail-Beziehungen einzurichten.

Element	Beschreibung
Vorhanden	Legt eine vorhandene, zuvor erstellte Datenmenge fest.
New	Legt eine neue Datenmenge fest.
Tabelle(n), die der Datenmenge hinzugefügt werden soll	Wählt eine Tabelle oder mehrere Tabellen aus, mit denen die Datenmenge gefüllt wird.

2.36 Datenbankkomponenten-Editor (Dialogfeld)

Im Dialogfeld **Datenbankkomponenten-Editor** definieren Sie die Verbindung, die zu einer Datenbank aufgebaut werden soll. Über die entsprechenden Eigenschaften können Sie den Typ der Datenbank, die Verbindungsparameter, den Benutzernamen, den SQLRole-Namen und das Passwort festlegen. Außerdem können Sie angeben, ob eine Aufforderung zur Anmeldung angezeigt werden soll.

Die Eigenschaften für die Datenbankkomponente können auch mithilfe des Objektinspektors festgelegt werden.

Doppelklicken Sie auf eine IBDATABASE-Komponente, um das Dialogfeld **Datenbankkomponenten-Editor** zu öffnen.

Dialogfeldoptionen

Verbindung

Option	Bedeutung
Lokal	Gibt an, dass sich die Datenbank auf dem lokalen Server befindet. Durch die Auswahl dieser Option wird die Schaltfläche Durchsuchen aktiviert, damit Sie die gewünschte Datenbank im Dialogfeld Datei öffnen angeben können.
Extern	Gibt an, dass sich die Datenbank auf einem Remote-Server befindet. Durch die Auswahl dieser Option werden die Felder Protokoll und Server aktiviert.
Protokoll	Legt das Protokoll fest, das für die Verbindung zum dem Remote-Server verwendet werden soll. Sie haben die Wahl zwischen TCP, NamedPipe und SPX.
Server	Der Name des Remote-Servers.
Datenbank	Der Name der Datenbank.

Datenbank-Parameter

Option	Bedeutung
Benutzername	Der Name des Datenbankbenutzers.
Passwort	Das Passwort des Datenbankbenutzers.
SQLRole	Der SQLRole-Name, der für die Datenbankverbindung verwendet wird.
Zeichensatz	Der Zeichensatz, der für die Datenbankverbindung verwendet wird.
Anmeldeaufforderung	Legt fest, ob für den Zugriff auf die Datenbank eine Anmeldung erforderlich ist.
Einstellungen	Zeigt die aktuellen Parameter an. Sie können weitere Parameter hinzufügen. Ein Beispiel: <code>user_name=sysdba password=masterkey sql_role_name=finance lc_ctype=WIN1252</code> Weitere Informationen über Datenbankparameter finden Sie im InterBase-API-Handbuch.

2.37 Transaktions-Editor (Dialogfeld)

Im Dialogfeld **Transaktions-Editor** werden die Transaktionsparameter festgelegt. Es werden vier Standardtransaktionseinstellungen angeboten, die Sie bei Bedarf anpassen können. Wenn Sie eine Standardtransaktion bearbeiten, wird das entsprechende Optionsfeld deaktiviert.

Eine vollständige Liste aller InterBase-Transaktionsparameter finden Sie im InterBase-API-Handbuch.

Die Eigenschaften für die Transaktionskomponente können auch mithilfe des Objektinspektors festgelegt werden.

Doppelklicken Sie auf eine IBTransaction-Komponente, um das Dialogfeld **Transaktions-Editor** zu öffnen. Die folgenden vier Optionen werden angezeigt:

Schnappschuss

Dieser Parameter ist standardmäßig auf concurrency und nowait eingestellt, d. h. die Transaktion erkennt andere Transaktionen und wartet nicht auf die Aufhebung von Sperren, sondern gibt einen Fehler zurück.

Committed lesen

Dieser Parameter ist standardmäßig auf read_committed, rec_version und nowait eingestellt, d. h. die Transaktion liest Änderungen, die von gleichzeitig ablaufenden Transaktionen vorgenommen werden, kann die zuletzt eingetragene Version einer Transaktion abrufen und wartet nicht auf die Aufhebung von Sperren, sondern gibt einen Fehler zurück.

Nur-Lesen Tabellenstabilität

Dieser Parameter ist standardmäßig auf read und consistency eingestellt, d. h. die Transaktion kann eine angegebene Tabelle lesen, und sie verhindert Änderungen durch andere Transaktionen.

Lesen-Schreiben Tabellenstabilität

Dieser Parameter ist standardmäßig auf write und consistency eingestellt, d. h. die Transaktion kann eine angegebene Transaktion lesen und in sie schreiben, und sie verhindert Änderungen durch andere Transaktionen.

Eine vollständige Liste aller InterBase-Transaktionsparameter finden Sie im InterBase-API-Handbuch.

2.38 IBUpdateSQL- und IBDataSet-Editor (Dialogfeld)

Verwenden Sie den Editor, um SQL-Anweisungen zur Aktualisierung einer Datenmenge zu erstellen.

Das TIBUpdateSQL-Objekt muss mit einem TIBQuery-Objekt verknüpft werden. Dazu weisen Sie der Eigenschaft UpdateObject des TIBQuery-Objekts den Namen des TIBUpdateSQL-Objekts zu, das die SQL-Anweisungen enthalten soll. Wählen Sie für das TIBQuery-Objekt einen Datenquellen- und Datenbanknamen. Außerdem müssen Sie in der Eigenschaft SQL eine SQL-Anweisung angeben, die eine Tabelle definiert.

So öffnen Sie den SQL-Editor:

1. Wählen Sie das TIBUpdateSQL- oder TIBDataSet-Objekt im Formular aus.
2. Klicken Sie mit der rechten Maustaste, und wählen Sie UpdateSQL-Editor oder DataSet-Editor.

Der UpdateSQL-Editor enthält die beiden Registerkarten **Optionen** und **SQL**.

Die Registerkarte Optionen

Die Seite **Optionen** wird beim ersten Aufruf des Editors angezeigt.

Tabellenname	Wählen Sie im Kombinationsfeld <i>Tabellenname</i> die zu aktualisierende Tabelle aus. Wenn Sie einen Tabellennamen angeben, werden die Listenfelder <i>Schlüsselfelder</i> und <i>Updatefelder</i> mit den verfügbaren Spalten gefüllt.
Schlüsselfelder	Das Listenfeld <i>Schlüsselfelder</i> wird genutzt, um die Spalten anzugeben, die während der Aktualisierung als Schlüssel verwendet werden sollen. Die gewählten Spalten sollten einem vorhandenen Index entsprechen.
Aktualisierungsfelder	Das Listenfeld <i>Updatefelder</i> gibt an, welche Spalten aktualisiert werden sollen. Wenn Sie eine Tabelle zum ersten Mal angeben, werden im Listenfeld <i>Updatefelder</i> alle Spalten ausgewählt. Wählen Sie die gewünschten Spalten aus.
Tabellenfelder lesen	Liest die Tabellenfelder der angegebenen Tabelle ein und listet diese auf..
Vorgabe Datenmenge	für Verwenden Sie diese Schaltfläche, um die Standardwerte der verknüpften Datenmenge wiederherzustellen. Danach sind alle Felder in den Listen Schlüsselfelder und Aktualisierungsfelder ausgewählt, und der alte Tabellenname wird wieder angezeigt.
Primärschlüssel auswählen	Klicken Sie auf diese Schaltfläche, um die Schlüsselfelder basierend auf dem Primärindex einer Tabelle auszuwählen.
SQL generieren	Wenn Sie eine Tabelle ausgewählt, die Schlüsselspalten angegeben und die zu aktualisierenden Spalten festgelegt haben, klicken Sie auf die Schaltfläche SQL generieren . Dadurch werden vorläufige SQL-Anweisungen generiert, die den Eigenschaften ModifySQL, InsertSQL, DeleteSQL und RefreshSQL der Aktualisierungskomponente zugeordnet werden können.
Bezeichner Anführungszeichen	in Wenn Sie dieses Kontrollkästchen aktivieren, werden alle Feldnamen in den generierten SQL-Anweisungen in Anführungszeichen eingeschlossen. Diese Option ist standardmäßig deaktiviert.

Die Registerkarte SQL

Wählen Sie die Registerkarte **SQL**, um die generierten SQL-Anweisungen zu betrachten und zu ändern. Wenn Sie diese Registerkarte nach der Erzeugung von SQL-Anweisungen aktivieren, enthält das Memofeld **SQL-Text** die Anweisung für die Eigenschaft ModifySQL. Sie können die Anweisung dann nach Bedarf ändern.

Anmerkung: Die generierten SQL-Anweisungen stellen lediglich das Grundgerüst für die Erzeugung von Aktualisierungsanweisungen dar. Meist müssen sie bearbeitet werden, um den gewünschten Effekt zu erzielen. Testen Sie jede einzelne Anweisung, bevor Sie sie akzeptieren.

Mit den Optionen der Optionsfeldgruppe **Typ der Anweisung** (Ändern, Einfügen, Löschen und Aktualisieren) können Sie zum gewünschten Anweisungstyp wechseln und die Anweisung bearbeiten.

Um die Anweisungen zu akzeptieren und sie den SQL-Eigenschaften der Aktualisierungskomponente zuzuordnen, klicken Sie auf *OK*.

2.39 Neue Verbindung

Verwenden Sie dieses Dialogfeld zur Erstellung einer neuen benannten Datenbankverbindungskonfiguration für dbExpress.

Element	Beschreibung
Treibername	Zeigt alle in der Datei dbxdrivers.ini aufgeführten Treiber an. Wählen Sie den gewünschten Treiber aus.
Name der Verbindung	Geben Sie hier einen eindeutigen Namen für die neue Verbindung ein. Der Name muss eindeutig sein, d.h. er darf nicht mit dem Namen einer anderen Verbindung in der Datei dbxconnections.ini übereinstimmen.

2.40 Neues Feld

Verwenden Sie dieses Dialogfeld, um in einer Datenmenge persistente Felder zu erstellen. Sie können persistente Felder hinzufügen oder vorhandene persistente Felder ersetzen.

Element	Beschreibung
Name	Tragen Sie hier den Namen der Komponente ein.
Komponente	Zeigt den Namen, den Sie im Feld Name angegeben haben, mit dem Komponentennamen als Präfix an. Das Produkt verwirft jede Eingabe, die Sie in dem Feld Komponente direkt vornehmen.
Typ	Wählen Sie den Datentyp der Feldkomponente aus. Sie müssen einen solchen Datentyp für jede von Ihnen neu erstellte Feldkomponente angeben. Um beispielsweise in einem Feld Fließkommawerte des Typs Currency anzuzeigen, wählen Sie aus der Dropdown-Liste den Eintrag Currency aus.
Größe	Legen Sie die maximale Anzahl der Zeichen fest, die in einem Stringfeld angezeigt oder eingegeben werden können, oder die Größe von Feldern des Typs Bytes und VarBytes. Für alle anderen Datentypen spielt die Größe keine Rolle.
Daten	Ersetzt ein vorhandenes Feld (z.B. um den Datentyp zu ändern) und basiert auf Spalten in der Tabelle oder auf einer Abfrage der zugrunde liegenden Datenmenge.
Berechnet	Zeigt Werte an, die von der OnCalcFields-Ereignisbehandlungsroutine einer Datenmenge zur Laufzeit berechnet werden.
Lookup	Übernimmt die Werte zur Laufzeit basierend auf festgelegten Suchkriterien aus einer angegebenen Datenmenge (unidirektionale Datenmengen unterstützen keine Nachschlagefelder).
InternalCalc	Ermittelt die in einer Client-Datenmenge gespeicherten berechneten Werte (die Werte werden also nicht dynamisch in einer OnCalcFields-Ereignisbehandlungsroutine berechnet). <i>InternalCalc</i> ist nur verfügbar, wenn Sie eine Client-Datenmenge bearbeiten. Die Werte eines InternalCalc-Felds sind Bestandteil der Client-Datenmenge.
Aggregat	Ermittelt den Wert eines Zusammenfassungsfelds aus Daten mehrerer Datensätze in einer Client-Datenmenge.
Schlüsselfelder	Führt die Felder in der aktuellen Datenmenge auf, für die die Werte übereinstimmen sollen. Um mehrere Felder festzulegen, geben Sie die Feldnamen direkt ein, anstatt sie aus der Dropdown-Liste auszuwählen. Trennen Sie die Feldnamen durch Semikolons. Für mehrere Felder müssen Sie persistente Feldkomponenten verwenden.
Datenmenge	Führt die Datenmengen auf, in welchen die Feldwerte nachgeschlagen werden sollen. Die Nachschlagedatenmenge darf nicht mit der Datenmenge für die Feldkomponente identisch sein, da sonst zur Laufzeit eine zirkuläre Referenz-Exception ausgelöst wird.
Schlüssel	Wird nur zum Erstellen von Nachschlagefeldern verwendet. Wenn Sie den Feldtyp Nachschlagen auswählen, sind im Gruppenfeld Nachschlage-Definition die Eingabefelder Schlüsselfelder und Datenmenge verfügbar. Die Felder Nachschlagefelder und Ergebnisfeld sind nur aktiv, wenn eine Verbindung zu einer zweiten Datenmenge besteht.
Ergebnisfeld	Führt die Felder in der Nachschlagedatenmenge auf, die als Wert des Nachschlagefeldes, das Sie erstellen, zurückgegeben werden können.

2.41 Beziehung

Verwenden Sie dieses Dialogfeld zum Definieren einer Beziehung zwischen Tabellen der Datenmenge.

Element	Beschreibung
Name	Legen Sie den Namen der Beziehung fest.
Übergeordnete Tabelle	Geben Sie die in der Beziehung übergeordnete Tabelle an.
Untergeordnete Tabelle	Legen Sie die in der Beziehung untergeordnete Tabelle fest.
Schlüsselspalten	Wählen Sie eine oder mehrere Spalten aus, die als Primärschlüssel in der übergeordneten Tabelle dienen sollen.
Fremdschlüsselspalten	Wählen Sie eine oder mehrere Spalten aus, die als Fremdschlüssel in der untergeordneten Tabelle dienen sollen. Diese Spalten müssen entweder bezüglich des Namens oder des Datentyps und Werts mit denjenigen korrespondieren, die Sie als Primärschlüssel ausgewählt haben.
Aktualisierungsregel	Wählen Sie die Regel zum Aktualisieren von Datensätzen aus. Bezieht sich auf Haupt/Detail-Beziehungen und darauf, wie Detaildatensätze aktualisiert werden, wenn ein Hauptdatensatz aktualisiert wurde.
Löschregel	Wählen Sie die Regel zum Löschen von Datensätzen aus. Bezieht sich auf Haupt/Detail-Beziehungen und darauf, wie Detaildatensätze gelöscht werden, wenn ein Hauptdatensatz gelöscht wurde.
Annehmen/Verwerfen-Regel	Wählen Sie die Annehmen/Verwerfen-Regel aus, die beim Einfügen in eine Haupt/Detail-Beziehung verwendet werden soll.

Siehe auch

[Standarddatenmengen verwenden](#)

2.42 Verbindung umbenennen

Geben Sie hier einen neuen Namen für die im Verbindungseditor ausgewählte benannte Verbindung ein. Die Datei dbxconnections.ini wird dadurch entsprechend aktualisiert. Der alte Name wird verworfen und kann nicht mehr für die Eigenschaft ConnectionName einer TSQLConnection-Komponente verwendet werden.

2.43 SortFields-Editor

Verwenden Sie dieses Dialogfeld, um die Felder für die Sortierung von Datensätzen einer SQL-Datensammlung festzulegen, deren Eigenschaft CommandType auf ctTable gesetzt ist.

Element	Beschreibung
Verfügbare Felder	Führt alle Felder der Datenbanktabelle auf. Wählen Sie gewünschte Felder aus, und verschieben Sie die Einträge mithilfe der Pfeilschaltflächen in die Liste Sortierung nach Feldern.
Sortierung Feldern nach	Zeigt die ausgewählten Felder an. Die Datensätze in der SQL-Datensammlung werden nach dem ersten Feld sortiert. Innerhalb von Gruppen, die das erste Feld definiert, werden die Datensätze nach dem zweiten Feld usw. sortiert.
Pfeilschalter	Mit diesen Schaltflächen können Sie die Felder zwischen den beiden Listen verschieben.

2.44 SQL-Monitor

Datenbank SQL-Monitor

Verwenden Sie dieses Dialogfeld zur Überwachung von aktuellen Anweisungsaufrufen, die über SQL Links an einen Remote-Server oder über das ODBC-Socket an eine ODBC-Datenquelle gesendet wurden.

Sie können verschiedene Aktivitätstypen überwachen, indem Sie mit **Optionen  Optionen für Ablaufverfolgung** das Dialogfeld Optionen für Ablaufverfolgung öffnen.

Optionen für Ablaufverfolgung - Seite Kategorien

Wählen Sie auf dieser Seite die Kategorien für die Ablaufverfolgung aus.

Element	Beschreibung
Vorbereitete Abfrage-Anweisungen	Vorbereitete Anweisungen, die an den Server gesendet werden sollen.
Ausgeführte Abfrage-Anweisungen	Anweisungen, die vom Server ausgeführt werden sollen. Beachten Sie bitte, dass eine einzelne Anweisung einmal vorbereitet und mit verschiedenen Parametern mehrmals ausgeführt werden kann.
Eingabeparameter	Parameterdaten, die bei INSERT- oder UPDATE-Anweisungen an den Server gesendet werden.
Abgerufene Daten	Daten, die von Servern ermittelt werden.
Anweisungsoperationen	Alle ausgeführten Operationen, wie z.B. ALLOCATE, PREPARE, EXECUTE und FETCH.
Verbinden/Trennen	Operationen, die im Zusammenhang mit dem Verbinden bzw. dem Aufheben einer Verbindung zu Datenbanken stehen, einschließlich dem Zuweisen von Verbindungs-Handle und der Freigabe von Verbindungs-Handle, wenn dies für den Server erforderlich ist.
Transaktionen	Transaktionsoperationen, wie z.B. BEGIN, COMMIT und ROLLBACK (ABORT).
Blob E/A	Operationen mit Blob-Datentypen, z.B. GET BLOB HANDLE, STORE BLOB usw.
Verschiedenes	Operationen, die nicht in anderen Kategorien enthalten sind.
Hersteller-Fehler	Vom Server zurückgegebene Fehlermeldungen. Die Fehlermeldung kann - abhängig vom Server - auch einen Fehlercode enthalten.
Hersteller-Aufrufe	API-Funktionsaufrufe an den Server. Beispielsweise ORLON für Oracle, ISC_ATTACH für InterBase.

Optionen für Ablaufverfolgung - Seite Puffer

Verwenden Sie diese Seite zum Verwalten des Ablaufverfolgungspuffers, der im Arbeitsspeicher vom SQL-Monitor unterhalten wird.

Element	Beschreibung
Puffergröße	Legt die Größe des Puffers für die Ablaufverfolgungsinformationen fest.
Zirkulär	Schreibt Ablaufverfolgungsinformationen in einen zirkulären Speicherpuffer, so dass bei Erreichen des Limits neuere Informationen die alten ersetzen.
Auf Festplatte	Schreibt Ablaufverfolgungsinformationen in eine Datei auf der Festplatte, wenn der Speicherpuffer voll ist.
Dateiname	Legt den Namen der Datei für die Option Auf Festplatte fest.

2.45 Stored Procedure (Dialogfeld)

Verwenden Sie dieses Dialogfeld, um eine Stored Procedure für ein BdpCommand-Objekt auszuwählen. Sie können Werte für Input- oder InputOutput-Parameter angeben und die ausgewählte Stored Procedure ausführen.

Element	Beschreibung
Stored Procedures	Zeigt alle Stored Procedures an, die für das dem BdpCommand-Objekt zugeordneten BdpConnection-Objekt verfügbar sind. Wenn Sie in dieser Dropdown-Liste eine Stored Procedure auswählen, werden in der Liste Parameter die Parameter für diese Stored Procedure angezeigt.
Stored Procedure hat eine oder mehr Ergebnismengen	Wenn die Stored Procedure einen oder mehrere Cursor zurückgibt, markieren Sie dieses Kontrollkästchen, damit die Ergebnismengen bei der Ausführung der Stored Procedure in dem Dialogfeld angezeigt werden.
Parameter	Zeigt alle Parameter für die aktuell ausgewählte Stored Procedure an. Wenn Sie in dieser Liste einen Parameter auswählen, werden die zugehörigen Metadaten im rechten Bereich angezeigt. Dort können Sie sie auch bearbeiten.
Execute	Versucht, die markierte Stored Procedure mit den aktuell ausgewählten Parametereinstellungen auszuführen. Die Ergebnisse der Stored Procedure (Output-Parameter, InputOutput-Parameter, Rückgabewerte, zurückgegebene Cursors) werden in das Datengitter im unteren Bereich des Dialogfeldes übernommen.

2.46 TableMappings Collection-Editor

Verwenden Sie dieses Dialogfeld, um Spalten Ihrer Datenquelle der im Arbeitsspeicher befindlichen Datenmenge zuzuordnen.

Element	Beschreibung
Anhand der Datenmenge Tabellen- und Spaltennamen vorschlagen	Zeigt eine Liste der verfügbaren, typisierten Datenmengen an, die vorhandene Tabellen- und Spaltennamen bereitstellt. Gibt nur eine Liste der Namen aus dem Schema der angegebenen, typisierten Datenmenge zurück. Wirkt sich nicht auf Standard-Datenmengen aus.
Datenmenge	Zeigt die Namen der Datenmengen an, die Sie für ein Modell verwenden können.
Quelltabelle	Zeigt den Namen der Datenquelltabelle an. Da ein Datenadapter mehr als eine Tabelle referenzieren kann, können Sie aus mehreren Tabellen in der Datenquelle auswählen.
Datenmengentabelle	Zeigt den Namen der Datenmengentabelle an. Da Sie Datenmengen erstellen können, die mehrere Tabellen enthalten, können hier mehrere Tabellen aufgeführt sein.
Quellspalten	Zeigt die Namen aller Spalten in der Datenquelltabelle an.
Datenmengenspalten	Zeigt die Namen der Spalten in der Datenmenge an, in die die Tabellenspalten der Datenquelle geschrieben werden sollen. Beim Aktualisieren der Datenquelle sind dies die Spalten, aus denen gelesen wird. Sie können die Namen ändern, um eine Spalte einer anderen, nicht korrespondierenden Spalte zuzuordnen oder Sie können korrespondierende Spalten zwischen der Quelltabelle der Datenmenge zuordnen.
Löschen	Löscht die aktiven Spaltennamen in den Quell- und Datenmengenlisten. Gelöschte Spalten werden in der Datenmenge nicht angezeigt. Verwenden Sie diese Option, wenn Ihre Abfrage mehr Spalten aus der Datenquelle zurückgibt, als für die Datenmenge benötigt werden.
Zurücksetzen	Setzt die Liste der Spaltennamen der Quelltabelle und der Datenmenge auf die ursprünglichen Werte zurück.
Eingabe-Taste	Sie können eine Zeile einfügen, indem Sie die Taste Eingabe drücken, während sich der Cursor in der letzten Zeile der Datenmengenspalte befindet. Dadurch können Sie neue Spalten anlegen, die bereits vorhanden sein könnten, aber aus bestimmten Gründen zur Entwurfszeit nicht angezeigt werden. Wenn Sie beispielsweise abgeleitete oder berechnete Felder haben, die Sie verfolgen möchten, könnten Sie eine neue Spalte für diese Daten hinzufügen. Die Reihenfolge der Spalten wirkt sich nicht auf die Daten aus.

2.47 Eindeutige Einschränkung

Verwenden Sie dieses Dialogfeld zum Auswählen der Datenmengenspalten, die in der Bedingung enthalten sein sollen. Diese Spalten können dann nur eindeutige Werte enthalten. Sie können auch eine oder mehrere Spalten als Primärschlüssel festlegen.

Element	Beschreibung
Name	Legt den Namen der Bedingung fest
Spalten	Markieren Sie alle Spalten, die in der Bedingung enthalten sein sollen. Wenn Sie eine nicht eindeutige Spalte als eindeutig angeben, könnten Sie einen Laufzeitfehler erzeugen.
Primärschlüssel	Indem Sie diese Option auswählen, legen Sie die markierten Spalten in der Bedingung als Primärschlüssel für das aktuelle DataTable-Objekt fest. Wenn Sie eine Kombination von Spalten festlegen, die auch Spalten mit doppelten oder Nullwerten enthält, könnten Sie einen Entwurfszeit- und einen Laufzeitfehler erzeugen. Bei einem Entwurfszeitfehler kann die Bedingung aber dennoch wie angegeben definiert bleiben, obwohl sie nicht korrekt ist.

2.48 Datenbankevolution

Dieses Dialogfeld wird eingeblendet, wenn Sie im EcoSpace-Designer oder im PersistenceMapperProvider-Designer auf die Schaltfläche Schema entwickeln klicken.

Element	Beschreibung
Registerseite Aktionen	Zeigt die auszuführenden Aktionen an, wenn das SQL-Script und das Zuordnungsinfo-Script für die Datenbank ausgeführt werden.
Warnungen	Diese Registerseite wird eingeblendet, wenn die Möglichkeit eines Datenverlustes besteht.
Fehler	Diese Registerseite wird eingeblendet, wenn Fehler auftreten, die die Evolution verhindern könnten.
SQL-Script	Zeigt die SQL-Anweisungen an, die bei der Evolution des Datenbankschemas ausgeführt werden.
Zuordnungsinfo-Anleitungen	Enthält Informationen über das Speichern der OR-Zuordnung in der Datenbank.
Zuordnungsinfo	Enthält die XML-Datei, die die neuen OR-Zuordnungsinformationen repräsentiert. Diese Informationen werden in der Datenbank gesichert und können auch als Datei gespeichert werden, indem Sie auf die Schaltfläche Script speichern klicken.
Script speichern	Klicken Sie diese Schaltfläche an, um das SQL-Script und die Zuordnungsinfo-XML-Datei zu speichern.
Execute	Führt das Evolutions-Script für die Datenbank aus.

Siehe auch

[Den EcoSpace-Designer verwenden](#)

[Den PersistenceMapperProvider-Designer verwenden](#)

[Benutzerdefinierte ECO-OR-Zuordnungsdateien](#)

2.49 EcoSpace-Designer

Der EcoSpace-Designer wird zur Konfiguration des Persistenzmechanismus benötigt, der in Ihrer Anwendung benutzt wird. Die nachstehenden Links bieten Informationen zum ECO-Framework und zur Verwendung des EcoSpace-Designer.

Siehe auch

- Überblick zum ECO-Framework
- Überblick zu ECO-Modellierungstools
- Den EcoSpace-Designer verwenden

2.50 Schema erzeugen

Dieses Dialogfeld wird eingeblendet, wenn Sie im EcoSpace-Designer oder im PersistenceMapperProvider-Designer auf die Schaltfläche Datenbankschema erzeugen klicken.

Element	Beschreibung
Registerseite Optionales Löschen	Diese Registerseite zeigt eine Liste der Tabellen an, die in der Datenbank vorhanden sind. Wählen Sie die Tabellen aus, die gelöscht werden sollen, indem Sie das Kontrollkästchen in der jeweiligen Listenzeile anklicken.
Entfernen/Neu erstellen erforderlich	Diese Registerseite zeigt eine Liste der in der Datenbank vorhandenen und ECO-relevanten Tabellen an. Die Tabellen in dieser Liste müssen entfernt und neu erstellt werden, damit alle darin enthaltenen Daten gelöscht werden.
Neue Tabellen	Diese Registerseite zeigt eine Liste der Tabellen an, die in der Datenbank erstellt werden.

Siehe auch

[Überblick zum ECO-Framework](#)

[Überblick zu ECO-Modellierungstools](#)

[Den EcoSpace-Designer verwenden](#)

[Den PersistenceMapperProvider-Designer verwenden](#)

2.51 OCL und ECO-AktionsspracheAusdruckseditor

Verwenden Sie dieses Dialogfeld, um OCL-Code und ECO-Aktionssprache Ausdrücke zu erstellen. Der Ausdrucksseditor ist ein Eigenschaftseditor, auf den sich vom Objektinspektor aus zugreifen lässt.

Element	Beschreibung
Bearbeitungsfeld Ausdruck	Zeigt den aktuellen Ausdruck an, der sich entweder manuell oder in vordefinierter Form aus dem Vervollständigungslistenfeld einfügen lässt.
Vervollständigungen	Zeigt eine Liste des OCL-Codes oder der ECO-Aktionssprache-Operatoren an, die im Kontext des aktuellen Ausdrucks gültig sind. Klicken Sie einen Eintrag in der Liste doppelt an, um ihn dem Ausdruck hinzuzufügen. Der ausgewählte Eintrag wird mit der korrekten Syntax zum Bilden eines gültigen OCL- oder ECO-Aktionssprache-Ausdrucks hinzugefügt.
Letzten entfernen	Entfernt das zuletzt zu dem Ausdruck hinzugefügte Attribut oder den zuletzt hinzugefügten Operator.
Löschen	Entfernt den gesamten Inhalt des Eingabefeldes OCL-Ausdruck, damit Sie von vorne beginnen können.
Register Parser-Meldung	In diesem schreibgeschützten Textfeld wird der aktuelle Status des Parsers angezeigt. Die Parser-Meldung wird aktualisiert, während Sie tippen oder Eigenschaften und Operatoren aus der Liste der gültigen Vervollständigungen hinzufügen.
Register Info	Zeigt eine Beschreibung der in der Liste der Vervollständigungen ausgewählten Operation an.
Register Ocl-Hilfe	Zeigt allgemeine Hilfeinformationen zu OCL an.
Wechsel zwischen Listenansicht und Baumdiagramm	Klicken Sie hierauf, um zwischen der Ansicht Baumdiagramm und dem Listenmodus umzuschalten. Der Standardmodus ist das Baumdiagramm.
Klassen hierarchisch anzeigen	Klicken Sie hierauf, um die Klassenvererbung in Form eines Baumdiagramms darzustellen.
Rückgabetypen anzeigen	Klicken Sie hierauf, um den Rückgabetyp aller OCL- und ECO-Aktionssprache-Operationen anzuzeigen.
Definierende Klasse anzeigen	Klicken Sie hierauf, um die definierende Klasse für die Attribute anzuzeigen.

Tip: Im Titel des Dialogfelds Ausdruckseditor wird der Kontext angezeigt, in dem der Ausdruck erstellt wird.

Anmerkung: Der Operator `->` bewirkt, dass die Ergebnisse des Ausdrucks links vom Operator als Kollektion behandelt werden. Wenn das Ergebnis keine Kollektion ist (z.B. nur ein einzelnes Element), wird es in eine Kollektion mit einem Element konvertiert.

Siehe auch

[Überblick zum ECO-Framework](#)

[Überblick zu ECO-Modellierungstools](#)

[Überblick zu Object Constraint Language \(OCL\)](#)

[verwenden ECO-Aktionssprache](#)

[ECO-Handles verwenden](#)

2.52 Packages für den EcoSpace auswählen

Verwenden Sie dieses Dialogfeld, um UML-Packages zu dem EcoSpace Ihrer Anwendung hinzuzufügen und daraus zu entfernen.

Element	Beschreibung
Ausgewählte UML-Packages	Zeigt eine Liste mit den ECO-UML-Packages, die aktuell in dem EcoSpace enthalten sind.
Verfügbare UML-Packages	Zeigt eine Liste mit den ECO-UML-Packages, die aktuell nicht in dem EcoSpace enthalten sind.
'<' Schalter	Verlagert das markierte ECO-UML-Package aus der Liste Verfügbare UML-Packages in die Liste Ausgewählte UML-Packages.
'>' Schalter	Verlagert das markierte ECO-UML-Package aus der Liste Ausgewählte UML-Packages in die Liste Verfügbare UML-Packages.
'<<' Schalter	Verlagert alle ECO-UML-Packages aus der Liste Verfügbare UML-Packages in die Liste Ausgewählte UML-Packages.
'>>' Schalter	Verlagert alle ECO-UML-Packages aus der Liste Ausgewählte UML-Packages in die Liste Verfügbare UML-Packages.

Anmerkung: Wenn Sie Änderungen an dem Modell vorgenommen haben, müssen Sie das Projekt erst neu erzeugen, bevor Sie den EcoSpace-Designer verwenden. Wenn die Liste Verfügbare UML-Packages unvollständig zu sein scheint, weist das darauf hin, dass das Projekt nicht neu erzeugt wurde.

Siehe auch

[Überblick zum ECO-Framework](#)

[Überblick zu ECO-Modellierungstools](#)

[Den EcoSpace-Designer verwenden](#)

2.53 Referenzierte ECO-Packages

Referenzierte ECO-Packages

Das Dialogfeld Referenzierte ECO-Packages ermöglicht es, externe ECO-Package-Dateien auszuwählen. Anschließend können Sie die Klassen in den ausgewählten ECO-Packages in Ihrem Projekt verwenden.

Element	Beschreibung
Liste der referenzierten Packages	Zeigt den Dateinamen und -pfad der ECO-Package-Datei an.
Hinzufügen	Ruft das Dialogfeld Datei öffnen auf. Verwenden Sie das Dialogfeld Datei öffnen, um zu der .ecopkg -Datei zu navigieren, auf die im Projekt referenziert werden soll.
Entfernen	Entfernt die aktuell ausgewählte Package-Datei aus der Liste.
Schließen	Schließt das Dialogfeld Referenzierte ECO-Packages.

Siehe auch

[Überblick zum ECO-Framework](#)

[Überblick zu ECO-Modellierungstools](#)

[Ein ECO-Package in einer DLL erstellen](#)

[Einer ECO-Package-DLL eine Referenz hinzufügen](#)

2.54 Experte zum Kapseln von Datenbanken

Sie können diesen Experten verwenden, indem Sie im ECO-Space-Designer auf das Tool Vorhandene Datenbank mit ECO versehen klicken.

Seite Zuordnungsdatei auswählen:

Element	Beschreibung
Pfad zur CodeGear ECO-XML-Datei	Wenn es eine vorhandene, benutzerdefinierte OR-Zuordnungsdatei gibt, die Sie für die Rückentwicklung der Datenbank benutzen möchten, können Sie den Dateinamen hier eingeben. Diese Angabe ist stets optional. Um mit der Anpassung der Datenbanktabellen fortzufahren, klicken Sie auf Weiter.
Durchsuchen...	Ruft das Dialogfeld Datei öffnen auf. Wählen Sie im Dialogfeld Datei öffnen die benutzerdefinierte OR-Zuordnungsdatei aus, die verwendet werden soll.
Package-Name	Mit diesem optionalen Feld können Sie den vorgeschlagenen Package-Namen überschreiben. Wenn der Experte keinen Package-Namen aus dem Datenbank-Verbindungsstring ermitteln kann, bleibt dieses Feld leer und Sie müssen den Package-Namen selbst eingeben.
Quelltext Klassenmodell erzeugen	<p>für Aktivieren Sie diese Option, damit der Experte die Quelltextdateien für die generierten Klassen erzeugt.</p> <p>Ist dieses Kontrollfeld nicht aktiv, erstellt der Experte das Modell und die Zuordnungsdatei. Sie müssen dann das Tool ECO-Quelltext neu erzeugen in der Modellansicht dazu verwenden, die Quelldateien für die Klassen im Modell zu erstellen.</p>

Seite Tabellen und Spalten auswählen:

Element	Beschreibung
Datenbank-Tabellenstruktur	<p>Das Diagramm zeigt die in der Datenbank gefundenen Tabellen an. Unter jeder Tabelle zeigt das Diagramm die zugehörigen Spalten an.</p> <p>Jeder Knoten verfügt über ein Kontrollfeld, mit dessen Hilfe Sie das Element in das generierte Modell und die Zuordnung einbeziehen oder ausschließen können.</p>
Knoteneigenschaften	<p>Die Eigenschaften des Knoten werden angezeigt, der im Datenbank-Tabellendiagramm ausgewählt ist.</p> <p>Bei Bedarf können Sie Änderungen an jedem Element vornehmen, um das generierte Modell und die Zuordnung anzupassen.</p>

Seite Klassen und Eigenschaften auswählen:

Element	Beschreibung
Klassen- und Eigenschaftendiagramm	<p>Das Diagramm zeigt die Klassen an, die aus den ausgewählten Tabellen und Tabellenspalten generiert werden. Unter jeder Klasse zeigt das Diagramm die Eigenschaften dieser Klasse an.</p> <p>Jeder Knoten verfügt über ein Kontrollfeld, mit dessen Hilfe Sie das Element in das generierte Modell und die Zuordnung einbeziehen oder ausschließen können.</p>
Knoteneigenschaften	<p>Die Eigenschaften des Knotens werden angezeigt, der im Klassen- und Eigenschaftendiagramm ausgewählt ist.</p> <p>Bei Bedarf können Sie Änderungen an jedem Element vornehmen, um das generierte Modell und die Zuordnung anzupassen.</p>

Siehe auch

[Benutzerdefinierte ECO-OR-Zuordnungsdateien](#)

[Modell und OR-Zuordnung aus einer vorhandenen Datenbank erstellen](#)

2.55 Ausrichtung

Bearbeiten▸Ausrichten

Mithilfe dieses Dialogfeldes können die markierten Komponenten relativ zueinander oder relativ zum Formular ausgerichtet werden.

Element	Beschreibung
Keine Veränderung	Ändert die Ausrichtung der Komponenten nicht.
Linke Seite	Richtet die ausgewählten Komponenten an ihren linken Rändern aus (nur horizontal).
Zentriert	Richtet die ausgewählten Komponenten an ihren Mittelpunkten aus.
Rechte Seite	Richtet die ausgewählten Komponenten an ihren rechten Rändern aus (nur horizontal).
Oben	Richtet die ausgewählten Komponenten an ihren oberen Rändern aus (nur vertikal).
Unten	Richtet die ausgewählten Komponenten an ihren unteren Rändern aus (nur vertikal).
Gleicher Abstand	Richtet die ausgewählten Komponenten mit gleichem Abstand zueinander aus.
In Fenster zentrieren	Richtet die ausgewählten Komponenten im Fenster zentriert aus.

2.56 Erstellungsfolge

[Bearbeiten](#) ▶ Erstellungsfolge

Verwenden Sie dieses Dialogfeld, um die Reihenfolge zu bestimmen, in der Ihr Programm die nicht-visuellen Komponenten erzeugt, wenn Sie das Formular während der Programmentwicklung oder zur Laufzeit laden.

Die Liste enthält die Namen und Typen der nicht-visuellen Komponenten des aktiven Formulars und deren Erstellungsfolge. Sie sind standardmäßig in der Reihenfolge aufgeführt, in der sie in das Formular eingefügt wurden.

2.57 Tabulator-Reihenfolge bearbeiten

[Bearbeiten](#) ▶ **Tabulator-Reihenfolge**

Verwenden Sie dieses Dialogfeld, um die Tabulator-Reihenfolge der Komponenten innerhalb des VCL-Formulars oder innerhalb der markierten Komponente (falls diese andere Komponenten enthält) zu ändern.

Die erste aufgeführte Komponente steht auch in der Tabulator-Reihenfolge an erster Position. Standardmäßig entspricht die Tabulator-Reihenfolge der Reihenfolge, in der Sie die Komponenten in das Formular eingefügt haben.

Klicken Sie den nach oben weisenden Pfeil an, um die Komponenten innerhalb der Tabulator-Reihenfolge nach oben (also nach vorn) zu verschieben. Für die entgegengesetzte Richtung klicken Sie den nach unten weisenden Pfeil an.

2.58 Skalierung

2

[Bearbeiten](#) ▶ Skalierung

Benutzen Sie dieses Dialogfeld, um alle Komponenten im aktuellen Formular um einen konstanten Faktor zu verkleinern oder zu vergrößern. Eine Skalierung des Formulars bewirkt neben einer Größenänderung der Komponenten auch eine Neupositionierung.

Geben Sie den Prozentwert ein, um den Sie die Größe der Komponenten des Formulars verändern wollen. Der Skalierungsfaktor muss zwischen 25 und 400 liegen. Prozentsätze über 100 vergrößern die Komponenten des Formulars. Prozentwerte unter 100 bewirken eine Verkleinerung der Formularkomponenten.

2.59 Größe

[Bearbeiten](#) ▶ Größe

Verwenden Sie dieses Dialogfeld zum Ändern der Größe mehrerer Komponenten, so dass sie die identische Höhe oder Breite haben.

Element	Beschreibung
Keine Veränderung	Ändert die Größe der Komponenten nicht.
Minimale Breite/Höhe	Weist den Steuerelementen die Höhe oder Breite der kleinsten ausgewählten Komponente zu.
Maximale Breite/Höhe	Weist den Steuerelementen die Höhe oder Breite der größten ausgewählten Komponente zu.
Breite	Legt eine bestimmte Breite für die markierten Komponenten fest.
Höhe	Legt eine bestimmte Höhe für die markierten Komponenten fest.

2.60 Fehleradresse nicht gefunden

Die Adresse, die Sie angegeben haben, konnte keiner Stelle im Quelltext zugewiesen werden. Dieser Fehler erscheint normalerweise immer dann, wenn einer der folgenden Punkte eingetreten ist:

- Die eingegebene Adresse ist falsch oder ist keine Adresse in Ihrer Anwendung.
- Das Modul, das diese Adresse beinhaltet, wurde nicht mit der Debugger-Information compiliert.
- Die Adresse gehört zu keiner Programmanweisung.

Denken Sie daran, dass Laufzeit- und Komponentenbibliotheken ohne Debugger-Informationen compiliert werden.

2.61 Eine andere Datei mit dem Namen <dateiname> befindet sich bereits im Suchpfad

Eine Datei mit dem gleichen Namen, den Sie gerade ausgewählt haben, ist bereits Bestandteil des Suchpfades.

2.62 Aufgrund des Hard-Mode konnte nicht gestoppt werden

Der integrierte Debugger hat entdeckt, dass Windows sich in einem modalen Zustand befindet, und dem Debugger nicht erlauben wird, Ihre Anwendung anzuhalten. Windows tritt in diesen Hard-Mode immer dann ein, wenn ein Zwischen-Task wie SendMessage ausgeführt wird, wenn keine Task-Warteschlange vorhanden ist, oder wenn das Menü System aktiv ist. Sie werden mit diesem Hard-Mode normalerweise nicht konfrontiert, es sei denn, Sie debuggen DDE- oder OLE-Prozesse in Delphi.

Ein eigenständiger Debugger, wie der Turbo Debugger für Windows, kann dazu verwendet werden, Anwendungen zu debuggen, sogar wenn Windows sich im Hard-Mode befindet.

2.63 Fehler beim Erstellen des Berichts: <prozess> (<fehlercode>)

Delphi war aus dem angegebenen Grund nicht in der Lage, die Anwendung zu starten.

Weitere Informationen über Fehler vom Typ "Zu wenig Arbeitsspeicher für die Ausführung" finden Sie in der Datei README.TXT.

2.64 Eine Komponentenklasse mit dem Namen <name> existiert bereits

Ein Package mit dem angegebenen Namen ist bereits in der IDE installiert. Benennen Sie das Package um, oder überprüfen Sie, ob das betreffende Package bereits vorhanden ist.

2.65 Ein Feld oder eine Methode mit dem Namen <name> existiert bereits

Der angegebene Name wird bereits von einer bestehenden Methode oder einem Feld verwendet.

Für eine vollständige Auflistung aller definierten Felder und Methoden prüfen Sie die Formulardeklaration am Anfang des Quelltextes der Unit.

2.66 Das Projekt enthält bereits ein Formular bzw. ein Modul mit dem Namen <name>

Jeder Modulname (Programm oder Bibliothek, Formular und Unit) in einem Projekt muss eindeutig sein.

2.67 Falsche Feld-Deklaration in der Klasse <klassenname>

Um das Formular und den Quelltext zu synchronisieren, muss Delphi in der Lage sein, die Deklaration jedes Feldes im ersten Abschnitt der Klassendeklaration des Formulars zu finden und zu bearbeiten. Obwohl der Compiler eine komplexe Syntax erlaubt, wird der Formular-Designer immer dann eine Fehlermeldung erzeugen, wenn nicht jedes Feld wie folgt deklariert ist:

```
type
  ...
  TForm1 = class(TForm)
    Field1:FieldType;
    Field2:FieldType;
  ...
  
```

Dieser Fehler tritt auf, weil mindestens eine Deklaration in diesem Abschnitt gelöscht, auskommentiert oder falsch geändert wurde. Benutzen Sie *Widerrufen*, um die Änderungen rückgängig zu machen, oder korrigieren Sie die Deklaration manuell.

Denken Sie daran, dass der Abschnitt mit der Klassendeklaration eines Formulars reserviert ist für den Gebrauch durch den Formular-Designer. Um Ihre eigenen Felder und Methoden zu deklarieren, platzieren Sie diese in **public**-, **private**- oder **protected**-Abschnitten.

2.68 Das Feld <feld> hat keine korrespondierende Komponente. Deklaration entfernen?

Der erste Abschnitt der Klassendeklaration Ihres Formulars definiert ein Feld, zu dem keine entsprechende Komponente im Formular existiert. Denken Sie daran, dass dieser Abschnitt für die Verwendung durch den Formular-Designer reserviert ist.

Um Ihre eigenen Felder und Methoden zu deklarieren, platzieren Sie diese in **public**-, **private**- oder **protected**-Abschnitten.

Dieser Fehler tritt immer dann auf, wenn Sie Binär-Formulardateien (.DFM) in den Quelltext-Editor laden, editieren und löschen, oder eine oder mehrere Komponenten umbenennen.

2.69 Das Feld <feld> sollte vom Typ <typ1> sein, ist jedoch als <typ2> deklariert. Deklaration korrigieren?

Der Typ des ausgewählten Feldes stimmt nicht mit seiner entsprechenden Komponente im Formular überein. Dieser Fehler tritt immer dann auf, wenn Sie die Felddeklaration im Quelltext-Editor ändern oder binäre Formulardateien (.DFM) in den Quelltexteditor laden und den Typ einer Komponente ändern.

Wenn Sie Nein auswählen und die Anwendung starten, tritt der Fehler auf, sobald das Formular geladen wird.

2.70 Deklaration der Klasse <klassenname> fehlt oder ist falsch

Delphi kann die Klassendeklaration des Formulars im Interface-Abschnitt der Unit nicht finden. Wahrscheinlich, weil die type-Deklaration, die die Klasse enthält gelöscht, auskommentiert oder nicht korrekt modifiziert wurde. Dieser Fehler tritt auf, wenn Delphi eine Klassendeklaration, die dem Folgenden entspricht, nicht finden kann:

```
type
  ...
  TForm1 = class(TForm)
  ...
  
```

Benutzen Sie Widerrufen, um Editierschritte rückgängig zu machen oder korrigieren Sie die Deklaration manuell.

2.71 Modulkopf fehlt oder ist fehlerhaft

Das Modul wurde gelöscht, auskommentiert oder falsch geändert. Benutzen Sie Widerrufen, um die Änderungen rückgängig zu machen, oder korrigieren Sie die Deklaration manuell.

Um Ihr Formular und den Quelltext zu synchronisieren, muss Delphi in der Lage sein, einen gültigen Modulkopf am Anfang des Quelltextes zu finden. Ein gültiger Modulkopf enthält das reservierte Wort `unit`, `program` oder `library`, gefolgt von einem Bezeichner (z.B. `Unit1`, `Project1`), gefolgt von einem Semikolon. Der Dateiname muss mit dem Bezeichner übereinstimmen.

Delphi sucht z.B. nach einer `Unit` names `Unit1` in `UNIT1.PAS`, einem Projekt namens `Project1` in `PROJECT1.DPR` und einer Bibliothek (DLL) namens `MyDLL` in `MYDLL.DPR`.

Beachten Sie, dass Modulbezeichner nicht länger als acht Zeichen sein dürfen.

2.72 IIMPLEMENTATION-Abschnitt fehlt oder ist falsch

Um Ihr Formular und den Quelltext zu synchronisieren, muss Delphi in der Lage sein, den Implementationsteil der Unit zu finden. Das reservierte Wort ist entweder gelöscht, auskommentiert oder falsch geschrieben worden.

Benutzen Sie den Befehl Widerrufen, um die Änderungen rückgängig zu machen, oder korrigieren Sie das reservierte Wort manuell.

2.73 Ungenügend Speicher zum Starten

Delphi war aufgrund unzureichenden Speicherplatzes oder unzureichender Windows-Ressourcen nicht in der Lage, Ihre Anwendung zu starten. Schließen Sie andere Windows-Anwendungen, und versuchen Sie es erneut.

Dieser Fehler tritt manchmal aufgrund von zu wenig Speicherplatz auf. Weitere Informationen finden Sie in der Datei README.TXT.

2.74 Haltepunkt wurde in eine Zeile gesetzt, die weder Quelltext noch Debug-Informationen enthält.

Ein Haltepunkt ist auf eine Quelltextzeile gesetzt, aus der kein Code generiert wird, oder in einem Modul, das nicht Teil dieses Projektes ist. Wenn Sie trotzdem starten, werden ungültige Haltepunkte deaktiviert (ignoriert).

2.75 <IDname> ist kein gültiger Bezeichner

Der Name des Bezeichners ist ungültig. Achten Sie darauf, dass das erste Zeichen ein Buchstabe oder ein Unterstrich (_) ist. Die nachfolgenden Zeichen können Buchstaben, Ziffern oder Unterstrich-Zeichen sein. Es dürfen sich außerdem keine Leerzeichen innerhalb des Bezeichners befinden.

2.76 Die Bibliothek <bibliothek> ist schon geladen, wahrscheinlich aufgrund eines ungültigen Programmabbruchs. Ihr System könnte instabil sein, deshalb sollten Sie Windows jetzt beenden und neu starten.

Beim Versuch, die Delphi-Komponentenbibliothek zu initialisieren, trat ein Fehler auf. Mindestens eine DLL ist, wahrscheinlich aufgrund einer falschen Programmbeendigung einer früheren Delphi- oder BDE-Sitzung, bereits im Speicher.

Sie sollten Windows beenden und neu starten.

2.77 Falsche Methoden-Deklaration in der Klasse <klassenname>

Um das Formular und den Quelltext zu synchronisieren, muss Delphi in der Lage sein, die Deklaration jeder Methode im ersten Abschnitt der Klassendeklaration des Formulars zu finden und zu bearbeiten. Der Formular-Designer wird immer dann eine Fehlermeldung erzeugen, wenn nicht jedes Feld wie folgt deklariert ist:

```
type
  ...
  TForm1 = class(TForm)
    Field1:FieldType;
    Field2:FieldType;
  ...
  <Method1 Declaration>;
  <Method2 Declaration>;
  ...
  ...
```

Dieser Fehler tritt auf, weil mindestens eine Methodendeklaration in diesem Abschnitt gelöscht, auskommentiert oder falsch geändert wurde. Benutzen Sie *Widerrufen*, um die Änderungen rückgängig zu machen, oder korrigieren Sie die Deklaration manuell.

Denken Sie daran, dass der Abschnitt mit der Klassendeklaration eines Formulars reserviert ist für den Gebrauch durch den Formular-Designer. Um Ihre eigenen Felder und Methoden zu deklarieren, platzieren Sie diese in **public**-, **private**- oder **protected**-Abschnitten.

2.78 Eine Implementation der Methode <methodenname> kann nicht gefunden werden

Die angezeigte Methode ist in der Klassendeklaration des Formular deklariert, aber kann nicht im Implementationsabschnitt der Unit gefunden werden. Sie ist möglicherweise gelöscht, auskommentiert, umbenannt oder falsch geändert worden.

Verwenden Sie den Befehl Widerrufen, um Änderungen rückgängig zu machen, oder korrigieren Sie die Prozedurdeklaration manuell. Stellen Sie aber sicher, dass die Deklaration in der Klasse identisch ist mit der in der Implementationssektion. (Dies wird automatisch erledigt, wenn Sie den Objektinspektor benutzen, um Ereignisbehandlungsroutinen zu erzeugen und umzubenennen.)

2.79 Die <methodenname>, aufgerufen von <formularname>.<ereignisname> hat eine inkompatible Parameterliste. Soll der Aufruf entfernt werden?

Ein Formular wurde geladen, das Eigenschaften enthält, die einer Methode mit einer inkompatiblen Parameterliste zugewiesen wurden. Parameterlisten sind inkompatibel, wenn die Anzahl der Parametertypen nicht übereinstimmt.

Um eine Liste der deklarierten Methoden, die kompatibel mit den Ereigniseigenschaften dieses Formulars sind abzurufen, müssen Sie die Dropdown-Liste auf der Seite Ereignisse des Objektinspektors auswählen.

Dieser Fehler tritt auf, wenn Sie eine Methodendeklaration manuell verändern, die von einer anderen Ereigniseigenschaft referenziert wird.

Beachten Sie, dass es nicht ratsam ist, dieses Programm auszuführen, ohne die Referenz zu entfernen oder den Fehler zu korrigieren.

2.80 Die Methode <methodenname>, aufgerufen von <formularname>, existiert nicht. Soll der Aufruf entfernt werden?

Die angezeigte Methode ist nicht mehr länger in der Klassendeklaration des Formulars vorhanden. Dieser Fehler tritt immer dann auf, wenn Sie eine Methode in der Klassendeklaration eines Formulars, das einem Ereignis zugewiesen wurde, manuell gelöscht oder umbenannt haben.

Wenn Sie Nein wählen und die Anwendung starten, tritt der Fehler beim Laden des Formulars auf.

2.81 Es wurde keine Anweisung für die aktuelle Zeile generiert

Sie versuchen, bis zur Cursor-Position auszuführen, haben jedoch eine Zeile ausgewählt, die keinen Code generiert hat, oder sich in einem Modul befindet, welches nicht zum Projekt gehört.

Geben Sie eine andere Zeile an und versuchen Sie es erneut.

Denken Sie daran, dass der intelligente Linker Prozeduren entfernt, die deklariert, aber nicht vom Programm aufgerufen werden (es sei denn, es sind virtuelle Methoden, die an ein eingelinktes Objekt gebunden sind).

2.82 Eigenschaft und Methode <methodenname> sind nicht kompatibel

Sie weisen eine Methode einer Ereigniseigenschaft zu, obwohl diese nicht kompatible Parameterlisten enthalten. Parameterlisten sind nicht kompatibel, wenn die Anzahl der Parametertypen nicht übereinstimmt. Um eine Liste von kompatiblen Methoden dieses Formulars abzurufen, wählen Sie die Dropdown-Liste auf der Seite Ereignisse des Objektinspektors aus.

2.83 <dateiname.PAS> oder <dateiname.DCU> kann im aktuellen Suchpfad nicht gefunden werden

Die gerade von Ihnen angegebene .pas- oder .dcu-Datei kann im aktuellen Suchpfad nicht gefunden werden.

Sie können diesen Suchpfad verändern, die Datei in diesen Suchpfad hineinkopieren, oder Sie löschen die Datei aus der Liste der installierten Units.

2.84 Source has been modified. Neu compilieren und binden?

Sie haben mindestens ein Quelltext- oder Formularmodul verändert, während Ihre Anwendung ausgeführt wurde. Wenn möglich, sollten Sie die Anwendung normal beenden (wählen Sie Nein, wechseln Sie in die laufende Anwendung, und wählen Sie im Systemmenü Beenden), und dann noch einmal starten oder neu compilieren.

Wenn Sie Ja auswählen, wird Ihre Anwendung geschlossen und anschließend erneut compiliert.

2.85 Symbol <symbol> wurde nicht gefunden.

Der Browser kann das angegebene Symbol nicht finden. Dieser Fehler tritt immer dann auf, wenn Sie ein falsches Symbol eingeben, oder wenn Debug-Informationen nicht für das das Symbol enthaltende Modul vorhanden ist.

2.86 Der Debugger läuft gerade. Beenden?

Ihre Anwendung läuft gerade und wird abgebrochen, wenn Sie fortfahren. Wenn möglich, sollten Sie dieses Dialogfeld abbrechen und Ihre Anwendung ganz normal beenden (z.B. durch Auswahl von Schließen im Systemmenü).

2.87 Die Uses-Klausel fehlt oder ist falsch.

Damit Ihre Formulare und der Quellcode synchron bleiben, muss Delphi die **uses**-Klausel für alle Module finden und warten können.

In einer Delphi-Unit muss eine **uses**-Klausel unmittelbar nach dem reservierten Wort `interface` stehen. In einem Programm oder einer Bibliothek muss eine **uses**-Klausel unmittelbar nach dem Programm- oder dem Bibliothekskopf stehen.

Dieser Fehler tritt auf, weil eine **uses**-Klausel gelöscht, auskommentiert oder nicht korrekt modifiziert wurde. Benutzen Sie *Widerrufen*, um die Änderungen rückgängig zu machen, oder korrigieren Sie die Deklaration manuell.

2.88 Ungültiges Ereignisprofil <name>

Das VBX-Steuerelement, das Sie installieren, ist ungültig.

2.89 Active Form-Experte

Datei ▶ Neu ▶ Weitere... ▶ Active Form

Mit dem Active Form-Experten fügen Sie einem ActiveX-Bibliotheksprojekt ein Active Form hinzu. Der Experte erstellt ein ActiveX-Bibliotheks-Projekt (falls erforderlich), eine Typbibliothek, ein Formular, eine Implementierungs-Unit und eine Unit, die entsprechende Typbibliotheks-Deklarationen enthält.

Element	Beschreibung
VCL-Klassenname	Bei Active Forms ist das Steuerelement deaktiviert, da diese immer auf TActiveForm basieren.
Neuer ActiveX-Name	Der Experte stellt einen Standardnamen bereit, über den Clients auf ihre ActiveX-Steuerelemente oder Active Forms zugreifen können. Ändern Sie diesen Namen, um einen anderen OLE-Klassennamen anzugeben.
Implementierungs-Unit	Der Experte vergibt einen Standardnamen für die Unit, die die CPP- und H-Dateien enthält, mit deren Code das Verhalten des Active Form implementiert wird. Sie können diesen Standardnamen übernehmen, oder einen neuen Namen eintippen.
Projektname	Active Forms müssen einem ActiveX-Bibliotheksprojekt hinzugefügt werden.. Wenn derzeit kein ActiveX-Bibliotheksprojekt geöffnet ist, können Sie in einem vierten Feld festlegen, welchem ActiveX-Bibliotheksprojekt das ActiveForm-Steuerelement hinzugefügt werden soll. Das Feld enthält bereits einen Standardnamen. Es ist deaktiviert, wenn bereits eine Active-X Bibliothek geöffnet ist.
Threading-Modell	Wählen Sie hier das Threading-Modell, das festlegt, wie COM Aufrufe an Ihr ActiveX-Formular serialisiert. Hinweis: Das gewählte Threading-Modell bestimmt, wie das Objekt registriert wird. Sie müssen sicherstellen, dass die Objektimplementierung dem gewählten Modell entspricht.
Versionsinformation hinzufügen	Mit dieser Option können Sie Versionsinformation in die OCX-Datei aufnehmen. Dadurch können Informationen über das Steuerelement angezeigt werden, (z.B. Copyright- und Datei-Beschreibungen) die im Browser ersichtlich sind. Geben Sie die gewünschten Informationen in der Registerkarte Versionsinfo des Dialogfensters Projektoptionen (Projekt ▶ Optionen) ein.
Info-Fenster hinzufügen	Mit diesem Kontrollkästchen können Sie ein Info-Fenster in das Projekt aufnehmen. Es handelt sich dabei um ein eigenes Formular, das beliebig bearbeitet werden kann. Das Info-Fenster enthält per Voreinstellung den Namen des Active Forms, ein Bild, Copyright-Informationen und eine OK-Schaltfläche.

2.90 Active-Server-Objekt-Experte

Datei ▶ Neu ▶ Weitere ▶ Active-Server-Objekte

Mithilfe des Active-Server-Objekt-Experten können Sie einfache Active-Server-Objekte erstellen. Sie müssen jedoch zuvor ein Projekt für eine Anwendung bzw. ActiveX-Bibliothek erstellen oder öffnen. Je nach Bedarf, kann das Projekt entweder eine Anwendung oder eine ActiveX-Bibliothek sein.

Im angezeigten Dialogfenster legen Sie die Eigenschaften des Active-Server-Objekts fest. Dieses spezielle Automatisierungsobjekt wird von dem Skript, das in einer Active Server-Seite ausgeführt wird, angelegt und aufgerufen.

Element	Beschreibung
Name der CoClass	Geben Sie den Namen des Objekts ein, das implementiert werden soll. (also den Namen der CoClass aus der Typbibliothek). Die erzeugte Implementierungsklasse hat denselben Namen mit einem vorangestellten 'T'.
Instantiiieren	Geben Sie einen Instantiierungs-Modus an, um festzulegen, wie Ihr Active-Server gestartet wird. (für In-Process-Server wird der Wert ignoriert).
Instantiiieren	Bedeutung
Intern	Das Objekt kann nur intern erzeugt werden. Externe Anwendungen können keine direkte Instantiierung durchführen.
Eine Instanz	Da bei diesem Typ für jede ausführbare Datei (Anwendung) nur eine COM-Schnittstelle möglich ist, führt das Erstellen mehrerer Instanzen zum Start mehrerer Anwendungen.
Mehrere Instanzen	Legt fest, dass mehrere Clients eine Verbindung zur Anwendung herstellen können. Immer wenn ein Client das Objekt anfordert, wird eine separate Instanz innerhalb eines eigenen Prozessraums angelegt (in einer ausführbaren Datei sind also mehrere Instanzen möglich).
Threading-Modell	Wählen Sie hier das Threading-Modell für das Objekt. Das gewählte Threading-Modell bestimmt, wie das Objekt registriert wird. Sie müssen sicherstellen, dass die Objektimplementierung dem gewählten Modell entspricht. Folgende Einstellungen können verwendet werden:
Modell	Beschreibung
Einfach	Es kann immer nur ein Client-Thread behandelt werden. COM serialisiert zu diesem Zweck alle eingehenden Aufrufe. Im Quelltext braucht keine Thread-Unterstützung implementiert zu werden.
Apartment	Auf jedes von einem Client instantiierte Objekt greift immer nur ein Thread zu. Sie müssen Schutzmaßnahmen implementieren, die verhindern, dass mehrere Threads auf den globalen Speicher zugreifen. Objekte können aber sicher auf ihre eigenen Instanzdaten (Eigenschaften und Elemente) zugreifen.
Frei	Jede Objektinstanz kann von mehreren Threads gleichzeitig aufgerufen werden. Sie müssen sowohl die Instanzdaten als auch den globalen Speicher schützen.
Beides	Dieses Modell entspricht dem vorhergehenden, stellt aber sicher, dass alle von Clients erstellten Callbacks in demselben Thread ausgeführt werden. Das bedeutet, dass Sie Werte, die als Parameter an Callback-Funktionen übergeben werden, nicht zu schützen brauchen.
Neutral	Mehrere Clients können das Objekt gleichzeitig in verschiedenen Threads aufrufen, aber COM stellt sicher, dass die Aufrufe nicht miteinander in Konflikt geraten. Sie müssen Thread-Konflikte verhindern, die globale Daten und Instanzdaten betreffen, auf die von mehreren Methoden zugegriffen wird. Dieses Modell sollte nicht bei Objekten mit Benutzeroberflächen verwendet werden. Das Modell steht nur unter COM+ zur Verfügung. Es wird unter COM auf das Apartment-Modell abgebildet.

Ereignismethoden auf Seitenebene(OnStartPage/ OnEndPage)	Im Objekt werden die Methoden OnStartPage und OnEndPage implementiert. Sie werden vom Web-Server beim Initialisieren und Finalisieren der Seiten aufgerufen. Diese Art von Active-Server-Objekten können mit IIS 3 und IIS 4 verwendet werden. Bei IIS 5 sollte die Option Objektkontext verwendet werden.
Objektkontext	Die Instanzdaten des Objekts werden mithilfe von MTS oder COM+ abgerufen. Diese Einstellung sollte für IIS 5 verwendet werden (sie ist aber auch bei IIS 4 und MTS möglich).
Für dieses Objekt ein Template-Test-Script generieren	Definieren Sie für den ersten unbekannten Begriff in einem Hilfethema einen Popup-Link (optional).
Element G	Es wird eine einfache ASP-Seite erstellt, die ein Active-Server-Objekt anhand seiner ProgID erzeugt. Sie können diese ASP-Seite so anpassen, dass die Methoden Ihres Objekts aufgerufen werden.

2.91 Hinzufügen

Verwenden Sie dieses Dialogfeld zum Hinzufügen eines Package zu der Requires-Klausel des aktuellen Package.

Element	Beschreibung
Paketname	Geben Sie den Namen des Package ein, das hinzugefügt werden soll. Wenn das Package im Suchpfad gefunden werden kann, ist keine vollständige Pfadangabe erforderlich. Wenn sich das betreffende Verzeichnis nicht im Suchpfad befindet, wird es am Ende des Suchpfades angefügt.
Suchpfad	Wenn Sie im Eingabefeld Name keinen vollständigen Verzeichnispfad angegeben haben, überprüfen Sie, ob das Verzeichnis mit dem Package in dieser Liste enthalten ist. Wenn Sie hier ein Verzeichnis in den Suchpfad einfügen, ändern Sie damit den globalen Bibliothekssuchpfad. Wenn ein Delphi-Package von einem anderen Package benötigt wird, muss der Compiler in der Lage sein, die zum Package gehörige .dcpl-Datei zu finden, damit kompiliert werden kann.

2.92 Experte für Automatisierungsobjekte

Datei ▶ **Neu** ▶ **Weitere...**

Verwenden Sie den Experten für Automatisierungsobjekte, um einem ActiveX-Bibliotheksprojekt einen Automatisierungsserver hinzuzufügen. Der Experte erzeugt eine Typbibliothek und die Definition für das Automatisierungsobjekt.

Element	Beschreibung
Name der CoClass	Geben Sie hier den Namen der Klasse an, deren Eigenschaften und Methoden den Client-Anwendungen zur Verfügung gestellt werden sollen (dem Namen wird automatisch ein T vorangestellt.)
Instantiiieren	Geben Sie einen Instantiiierungs-Modus an, um festzulegen, wie Ihr Automatisierungsserver gestartet wird.
Threading-Modell	Wählen Sie hier das Threading-Modell, um festzulegen, wie COM die Aufrufe der Automatisierungsobjekt-Schnittstelle serialisiert. Das gewählte Threading-Modell bestimmt, wie das Objekt registriert wird. Sie müssen sicherstellen, dass die Objektimplementierung dem gewählten Modell entspricht.
Ereignisunterstützung generieren	Mithilfe dieser Option kann eine eigene Schnittstelle für die Verwaltung der Ereignisse des Automatisierungsobjekts implementiert werden. Diese Schnittstelle hat den Namen <i>ICoClassNameEvents</i> und definiert die Ereignisbehandlungsroutinen, die vom Client implementiert werden müssen. Die Schnittstelle wird nicht von der Anwendung implementiert.

Die Dropdown-Liste Instantiierung kann die folgenden Instantiierungstypen enthalten:

Instantiieren	Bedeutung
Intern	Das Objekt kann nur intern erzeugt werden. Externe Anwendungen können keine direkte Instantiierung durchführen.
Eine Instanz	Da bei diesem Typ für jede ausführbare Datei (Anwendung) nur eine COM-Schnittstelle möglich ist, führt das Erstellen mehrerer Instanzen zum Start mehrerer Anwendungen.
Mehrere Instanzen	Legt fest, dass mehrere Clients eine Verbindung zur Anwendung herstellen können. Immer wenn ein Client das Objekt anfordert, wird eine separate Instanz innerhalb eines eigenen Prozessraums angelegt (in einer ausführbaren Datei sind also mehrere Instanzen möglich).

Anmerkung: Unter COM+ hängt die Serialisierung der Objektaufrufe auch von der Art und Weise ab, in der das Objekt an Aktivitäten beteiligt ist. Die entsprechenden Einstellungen können in der Registerkarte **COM+** des Typbibliothekseditors oder mit dem Komponentenmanager von COM+ vorgenommen werden.

Die Dropdown-Liste Threading-Modell kann die folgenden Instantiierungstypen enthalten:

Threading-Modell	Bedeutung
Single Apartment	Es kann immer nur ein Client-Thread behandelt werden. COM serialisiert zu diesem Zweck alle eingehenden Aufrufe. Im Quelltext braucht keine Thread-Unterstützung implementiert zu werden.
Frei	Jede Objektinstanz kann von mehreren Threads gleichzeitig aufgerufen werden. Sie müssen sowohl die Instanzdaten als auch den globalen Speicher schützen.
Both	Dieses Modell entspricht dem vorhergehenden, stellt aber sicher, dass alle von Clients erstellten Callbacks in demselben Thread ausgeführt werden. Das bedeutet, dass Sie Werte, die als Parameter an Callback-Funktionen übergeben werden, nicht zu schützen brauchen.

Neutral	Mehrere Clients können das Objekt gleichzeitig in verschiedenen Threads aufrufen, aber COM stellt sicher, dass die Aufrufe nicht miteinander in Konflikt geraten. Sie müssen Thread-Konflikte verhindern, die globale Daten und Instanzendaten betreffen, auf die von mehreren Methoden zugegriffen wird. Dieses Modell sollte nicht bei Objekten mit Benutzeroberflächen verwendet werden. Das Modell steht nur unter COM+ zur Verfügung. Es wird unter COM auf das Apartment-Modell abgebildet.
---------	---

Anmerkung: Unter COM+ hängt die Serialisierung der Objektaufrufe auch von der Art und Weise ab, in der das Objekt an Aktivitäten beteiligt ist. Die entsprechenden Einstellungen können in der Registerkarte COM+ des Typbibliotheksseditors oder mit dem Komponentenmanager von COM+ vorgenommen werden.

2.93 COM-Objekt-Experte

Datei ▶ Neu ▶ Weitere...

Mithilfe des COM-Objekt-Experten können Sie einfache COM-Objekte wie beispielsweise eine Shell-Erweiterung erstellen. Sie müssen jedoch zuvor ein Projekt für eine Anwendung bzw. ActiveX-Bibliothek erstellen oder öffnen. Je nach Bedarf, kann das Projekt entweder eine Anwendung oder eine ActiveX-Bibliothek sein.

Element	Beschreibung
Name der CoClass	Geben Sie hier den Namen der Klasse an, deren Eigenschaften und Methoden den Client-Anwendungen zur Verfügung gestellt werden sollen (also den Namen der CoClass). Dem Namen der Implementierungsklasse wird automatisch ein T vorangestellt.
Instantiieren	Geben Sie den Instantiierungs-Modus an, der festlegt, wie Ihr Automatisierungs-Server gestartet wird. Wenn Sie das COM-Objekt als prozessinternen Server verwenden, wird diese Option ignoriert.
Threading-Modell	Wählen Sie das Threading-Modell, um anzugeben, wie Client-Anwendungen Ihre COM-Objekt-Schnittstelle aufrufen können. Das gewählte Threading-Modell bestimmt, wie das Objekt registriert wird. Sie müssen sicherstellen, dass die Objektimplementierung dem gewählten Modell entspricht.
Implementiertes Interface	Geben Sie hier den Namen des Standard-Interface für das COM-Objekt an. Standardmäßig ist dies der Name der CoClass mit vorangestelltem I. Wenn Sie das Standard-Interface übernehmen, erhalten die Objekte ein neues Interface, das von <i>IUnknown</i> abgeleitet ist. Sie können das Interface dann mit dem Typbibliothekseditor definieren. Um den Standardnamen zu ändern, geben Sie einen neuen Namen in das Textfeld ein. Anstatt ein neues Interface zu implementieren, können Sie auch festlegen, dass das Objekt ein duales oder benutzerdefiniertes Interface aus einer im System registrierten Typbibliothek implementiert. Um dieses Interface auszuwählen, klicken Sie auf die Schaltfläche Liste. Dadurch wird der Experte für Interface-Auswahl geöffnet. Das Laden des Experten dauert einige Zeit, da er alle Interfaces ausfindig machen muss, die in den im System registrierten Typbibliotheken definiert sind. Beachten Sie, dass Sie den Experten verwenden müssen, um ein vorhandenes Interface zu implementieren. Wenn Sie den Namen eines vorhandenen Interface eingeben, geht der Experte davon aus, dass sie dem Objekt nur einen anderen Namen zuweisen wollen.
Beschreibung	Geben Sie eine Beschreibung des COM-Objekts ein.
Typbibliothek einschließen	Aktivieren Sie diese Option, um eine Typbibliothek für das Objekt zu generieren. Mithilfe dieser Bibliothek können Client-Anwendungen Informationen über Objekt-Interfaces und deren Eigenschaften und Methoden abrufen.
Interface Ole-Automatisierung als	Aktivieren Sie diese Option, um Typbibliothek-Marshaling zu ermöglichen. Sie brauchen dann keine eigene Proxy-Stub-DLL für benutzerdefiniertes Marshaling zu erstellen. Bei dieser Option müssen Sie sicherstellen, dass das Interface Typen verwendet, die zur OLE-Automatisierung kompatibel sind.

2.94 Dialogfenster für COM+ Ereignisschnittstelle-Auswahl

Klicken Sie im COM+-Subskriptionsobjekt-Experten auf die Schaltfläche Durchsuchen, um das Dialogfenster für die COM+ Ereignisschnittstelle-Auswahl zu öffnen. Danach werden alle Ereignisklassen aufgelistet, die aktuell in dem COM+-Katalog installiert sind. Verwenden Sie die im Dialogfenster enthaltene Durchsuchen Schaltfläche, um eine Typbibliothek, die die Ereignisschnittstelle enthält zu suchen und zu wählen.

2.95 Experte für COM+ Ereignisobjekte

Datei ▶ **Neu** ▶ **Weitere...**

Der Experte für COM+ Ereignisobjekte erstellt ein COM+ Ereignisobjekt, das von einem transaktionalen Server aufgerufen werden kann, um Ereignisse auf Clients zu generieren. Da das Projekt für ein COM+ Objekt nur weitere COM+ Objekte enthalten kann, werden Sie beim Start des Experten möglicherweise aufgefordert, ein neues Projekt anzulegen.

Element	Beschreibung
Name der CoClass	Geben Sie den Namen des COM+ Ereignisobjekts an. Server-Objekte, die COM+ Ereignisse generieren, erstellen eine Instanz dieses Objekts und rufen ihre Ereignisse auf. Die Ereignisse werden von COM+ so verteilt, dass sie auf registrierten Clients ausgelöst werden.
Interface	Geben Sie den Namen des Interface an, das Ereignisbehandlungsroutine für alle Ereignisse definiert, die vom COM+ Ereignisobjekt verwaltet werden. Da das Interface von Client-Ereigniszielden implementiert wird, generiert der Experte keine Implementierungs-Unit.
Beschreibung	Geben Sie hier (optional) eine kurze Beschreibung der Ereignisobjekte ein, an der die Clients den Zweck der Ereignisse erkennen.

Wenn die Aktionen im Experten abgeschlossen sind, können Sie die Methoden des generierten Interface mit dem Typbibliothekseditor definieren. Bei der Definition dieses Interface gelten folgende Regeln:

- Alle Methodennamen müssen für alle Schnittstellen des Ereignisobjekts eindeutig sein.
- Alle Methoden müssen einen HRESULT-Wert zurückgeben.
- Der Modifizierer für alle Methodenparameter muss leer sein.

2.96 COM+-Subskriptionsobjekt-Experte

Datei ▶ **Neu** ▶ **Weitere** ▶ **COM+-Subskriptionsobjekt**

Erstellt eine COM+-Ereignissubskriptionskomponente mit dem COM+-Subskriptionsobjekt-Experten. Diese Komponente setzen Sie mit einem COM+-Ereignisobjekt ein, um Benachrichtigungen über von COM+-Publisher-Anwendungen ausgelöste Ereignisse zu erhalten.

Element	Beschreibung
Klassenname	Geben Sie den Namen der Klasse ein, die das Ereignis-Interface implementiert.
Threading-Modell	Wählen Sie ein Threading-Modell aus. Das Threading-Modell für eine Komponente bestimmt, wie den Threads zugewiesene Methoden ausgeführt werden.
Interface	In dieses Feld können Sie den Namen der Schnittstelle eingeben, oder verwenden Sie die Schaltfläche Durchsuchen, um alle Ereignisklassen aufzulisten, die in dem COM+-Katalog installiert sind. Es wird ein Auswahlfenster geöffnet, in dem sich auch eine Durchsuchen-Schaltfläche befindet. Verwenden Sie diese, um eine Typbibliothek, die das Ereignis-Interface enthält zu suchen und zu wählen.
Existierendes Interface implementieren	Wenn Sie ein Interface auswählen, gibt Ihnen der Experte die Möglichkeit, das von der Ereignisklasse unterstützte Interface automatisch zu implementieren. Wenn Sie das Kontrollkästchen <i>Existierendes Interface implementieren</i> aktivieren, übernimmt der Experte für Sie automatisch das Stubbing-out der einzelnen Methoden in das Interface.
Vorfahr-Interfaces implementieren	Wenn Sie ein Interface auswählen, gibt Ihnen der Experte die Möglichkeit, die von der Ereignisklasse unterstützten Vorfahr-Interfaces automatisch zu implementieren. Wenn Sie möchten, dass der Experte vererbte Interfaces implementiert, aktivieren Sie das Kontrollkästchen <i>Vorfahr-Interfaces implementieren</i> . Folgende drei Vorfahr-Interfaces können nicht vom Experten implementiert werden: IUnknown, IDispatch und IAppServer.
Beschreibung	Geben Sie hier eine kurze Beschreibung Ihrer Ereignissubskriptionskomponente ein.

2.97 Neues Menü anpassen

Datei>Neu>Anpassen

Verwenden Sie dieses Dialogfeld, um den Inhalt des Menüs Datei>Neu anzupassen, indem Sie die Menüelemente aus dem mittleren in den rechten Bereich ziehen.

Element	Beschreibung
Galerie-Elemente (linker Bereich)	Zeigt die Ordner der Galerie-Elemente an, die in der Objektablage zur Verfügung stehen. Klicken Sie auf einen Ordner, um dessen Inhalt im mittleren Bereich anzuzeigen.
Menüelemente (rechter Bereich)	Zeigt die Elemente an, die aktuell im Menü Datei>Neu aufgeführt sind. Um ein Element aus dem Menü Datei>Neu zu entfernen, ziehen Sie es aus der Liste, bis das Symbol X erscheint. Lassen Sie dann die Maustaste los. Zum Ändern des Textes eines Menüelements doppelklicken Sie auf den Text und geben den neuen Text ein. Um eine Trennlinie zwischen Menüelementen einzufügen, ziehen Sie die Trennlinie aus dem mittleren Bereich in die Menüliste.
Ziehen Sie ein Element hierher	Wenn Sie einen Standardanwendungstyp festlegen möchten, ziehen Sie das Element, das den gewünschten Anwendungstyp repräsentiert, aus dem mittleren Bereich und legen es auf dieser Schaltfläche ab. Zum Entfernen der Standardanwendung klicken Sie auf die Schaltfläche.

2.98 Namen der Zielfile ändern

Verwenden Sie dieses Dialogfeld, um die ausgewählte Datei beim Kopieren in das Zielverzeichnis zu ändern. Die Umbenennung erfolgt nur, wenn Sie die Datei in das Zielverzeichnis kopieren. Die Quelldatei wird nicht umbenannt.

Wenn die Datei mit dem ursprünglichen Namen bereits im Zielverzeichnis vorhanden ist, verbleibt sie dort so lange, bis sie gelöscht wird.

Element	Beschreibung
Name der Quelldatei	Gibt den Namen der Datei im Quellverzeichnis an.
Unterverzeichnis	Gibt den Namen des Unterverzeichnisses an, in dem sich die Quelldatei befindet.
Name der Zielfile	Geben Sie den neuen Namen für die Zielfile ein.

Siehe auch

[Deployment von ASP.NET-Anwendungen](#)

[Den ASP.NET-Deploymentmanager verwenden](#)

2.99 FTP-Verbindungsoptionen

Verwenden Sie dieses Dialogfeld, um für den Deploymentmanager FTP-Server-Verbindungsinformationen festzulegen.

Element	Beschreibung
Server	Geben Sie den Internet-Host-Namen oder die IP-Adresse für den FTP-Server an. Das Präfix FTP:// wird automatisch vorangestellt..
Port	Gibt den Port an, der auf dem FTP-Server für die Verbindung verwendet werden soll. Der Standard-Port ist 21.
Passiver Modus	Veranlasst, dass die Verbindung von Ihrem Computer und nicht von dem FTP-Server eingerichtet wird. Markieren Sie Passiver Modus, wenn Ihr Computer durch eine Firewall geschützt ist, die eine vom FTP-Server initiierte Verbindung blockieren würde.
Ordner	Legt das Verzeichnis auf dem FTP-Server fest, in das Sie Dateien kopieren möchten.
Anonymes Login	Meldet sich mit dem Benutzernamen anonymous und Ihrer Email-Adresse als Passwort bei dem FTP-Server an, ohne ein tatsächliches Konto auf dem Server zu verwenden. (Der FTP-Server muss für anonymous Logins konfiguriert sein.)
E-Mail	Wenn Sie Anonymes Login markiert haben, geben Sie hier Ihre Email-Adresse ein, z.B. YourName@domain.com .
Benutzername	Legt den Benutzernamen für die Verbindung fest, wenn kein anonymes Login verwendet wird.
Passwort	Legt das Passwort für die Verbindung fest, wenn kein anonymes Login verwendet wird.
Passwort (als Text) speichern	Speichert das Passwort in der Deploymentmanager-Datei (.bdsdeploy) im Projektverzeichnis als nicht verschlüsselten Text und stellt somit ein Sicherheitsrisiko dar.
Test	Testet die Verbindung zu dem FTP-Server und liefert ein Feedback.

Siehe auch

Deployment von ASP.NET-Anwendungen

Den ASP.NET-Deploymentmanager verwenden

2.100 Deploymentmanager

Wählen Sie in diesem Dialogfeld die Dateien aus, die Sie als Teil Ihrer ASP.NET-Anwendung weitergeben möchten.

Siehe auch

[Deployment von ASP.NET-Anwendungen](#)

[Den ASP.NET-Deploymentmanager verwenden](#)

2.101 Von RAD Studio erzeugte Dateien

Die folgende Tabelle enthält die Erweiterungen der von RAD Studio erzeugten Dateien.

Anmerkung: Für MSBuild müssen die Erweiterungen aller Projektdateien mit 'proj' enden (d.h., MSBuild verwendet die Maske `*.*proj`).

Dateierweiterung	Beschreibung
app.config	Dynamische Eigenschaften und deren Werte für .NET Windows-Anwendungen.
asax, ascx, asmx, aspx	ASP.NET-Anwendungsdateien.
bdsproj	Projektoptionsdatei für BDS 2006 und früher. Enthält die aktuellen Einstellungen für Projektoptionen, wie z.B. Compiler- und Linker-Einstellungen, Verzeichnisse, bedingte Direktiven und Befehlszeilenparameter. Setzen Sie die Optionen mit dem Menübefehl Projekt > Optionen .
bdsgroup	Projektgruppen für BDS 2006 und früher.
bpk	Quelltextdatei für ein C++ Builder-Package; erzeugt beim Compilieren und Linken eine .bpl-Datei.
bpl	Ein compiliertes Delphi-Package oder ein compiliertes C++-Package (siehe auch .dpk).
bpr	C++Builder-Projektquelltextdatei; erzeugt beim Compilieren eine .exe-, .dll- oder .ocx-Datei.
cbproj	C++-Projekt. Nicht zu verwechseln mit 'cproj', das ein VS.NET C++-Projekt ist.
cfg	Projektkonfigurationsdatei, die beim Compilieren von der Befehlszeile aus verwendet wird. Der Compiler sucht im Compiler-Ausführungsverzeichnis nach der Datei dcc32.cfg , dann im aktuellen Verzeichnis nach der Datei dcc32.cfg und schließlich im Projektverzeichnis nach der Datei projektnname.cfg . Sie können in der Befehlszeile dcc32 projektnname eingeben, und das Projekt mit denselben Optionen wie in der IDE festgelegt compilieren.
config	Config-Dateien enthalten Projektoptionsinformationen und die Build-Logik. Zu jedem Projekt gehört eine .config-Datei.
cpp	C++-Quelltextdatei
cs	Die Codebehind-Datei für ASP.NET; auch eine C#-Quelltextdatei.
csm	Vorcompilierte C++-Header-Datei.
csproj	C#-Projekt, in Übereinstimmung mit MSBuild-Konventionen.
dci	Enthält die Änderungen von Code Insight, die in der IDE vorgenommen wurden.
dcp	Enthält alle Compilier- und Link-Informationen für ein Delphi-Package, vergleichbar mit einer .dcu-Datei, die diese Informationen für eine .pas-Datei enthält. Verwenden Sie diese Datei, wenn Sie Anwendungen mit Laufzeit-Packages erzeugen.
dcpil	Compilierter Delphi.NET CLR-Import. Entspricht der .dcp-Datei für .NET.
dcu	Compilierte Delphi-Unit. Eine Zwischenausgabedatei des Compilers, die für jeden Quelltext einer Win32-Unit erzeugt wird. Sie müssen diese binären Dateien nicht öffnen oder mit Ihrer Anwendung weitergeben. Die Datei .dcu aktiviert das schnelle Compilieren und Linken.
dcuil	Eine Zwischenausgabedatei des Compilers, die für jeden Quelltext einer .NET-Unit erzeugt wird. Entspricht der .dcu-Datei für .NET.
delphi.dct	Änderungen an der Komponenten-Template, die Sie in der IDE vorgenommen haben.
dfm	Eine Windows VCL-Formulardatei.

dpk	Die Quelltextdatei für ein Delphi-Package. Erzeugt beim Compilieren eine .bpl -Datei.
dpr	Delphi-Projektquelltextdatei; erzeugt beim Compilieren eine .exe-, .dll- oder .ocx-Datei.
dproj	Delphi-Projekt (nativ und .NET).
dsk	Datei zum Speichern des Desktops, wenn die Option "Projekt-Desktop automatisch speichern" aktiviert ist.
dst	Datei zum Speichern der Desktop-Schnelleinstellung, die in dem Kombinationsfeld "Desktop" in der IDE-Symbolleiste festgelegt ist.
exe	Ausführbare Datei.
exe.incr	Inkrementelle Build-Informationen.
groupproj	Projektgruppe.
h	C++-Header-Quelltextdatei.
hpp	Von Pascal erzeugte C++-Header-Datei.
i	C++-Präprozessorausgabe (wird nicht standardmäßig gespeichert). Jede .cpp-Datei und alle eingeschlossenen Header werden vom Präprozessor in eine .i-Datei eingefügt.
identcache	Informationen, die für das Refactoring verwendet werden.
il?	Inkrementelle C++Link-Zustandsdatei.
nfm	Eine .NET VCL-Formulardatei.
nfn	Eine Datei, die von den Übersetzungs-Tools verwaltet wird. Sie enthält übersetzte Strings und andere Daten, die im Translation-Manager angezeigt werden. Für jedes Formular in der Anwendung und für jede Zielsprache wird eine separate .nfn -Datei verwaltet.
obj	Compilierte C++-Übersetzungs-Unit. Jede .cpp-Datei und alle eingeschlossenen Header werden in eine .obj-Datei kompiliert.
optset	Benannte Optionsgruppdatei, die für jedes Projekt Konfigurationsoptionen speichert.
pas	Die Delphi-(Pascal)-Quelltextdatei.
pdb	Enthält Debugging-Informationen des Projekts für .NET.
res, rc	Compilierte und nicht compilierter Ressourcendateien.
resources	Eine binäre Ressourcendatei, die in eine ausführbare Laufzeitdatei eingebettet oder in eine Satellite-Assemblierung für .NET kompiliert werden kann.
resx	Eine XML-Ressourcendatei, die Knoten enthält, die Objekte und Strings für .NET repräsentieren.
rsp	Antwortdatei, die vom C++-Linker verwendet wird.
targets	Targets-Datei, eine MSBuild-konforme XML-Datei, die Sie Ihrem Projekt hinzufügen, damit der Build-Prozess angepasst werden kann. Sie enthält MSBuild-Skripts und andere Informationen.
tlb	Typbibliothek.
todo	Die To-Do-Liste des Projekts.
tvsconfig	Konfigurationsdatei für die Modellierung.
txvpck, txvcls	Informationen für Modelldiagramme.
vbproj	Visual Basic-Projekte, in Übereinstimmung mit den VS.NET- und MSBuild-Konventionen.
web.config	Dynamische Eigenschaften und deren Werte für ASP.NET-Anwendungen.
#nn	#nn = #00, #01, #02, etc. vorcompilierte C++-Header-Datei.

2.102 Experte für Interface-Auswahl

Datei ▶ **Neu** ▶ **Weitere...** ▶ **Neues COM-Objekt**

Auf den Experten für Interface-Auswahl greifen Sie über die Schaltfläche Liste des COM-Objekt-Expertens zu. In dem Experten für Interface-Auswahl können Sie ein vordefiniertes duales oder ein benutzerdefiniertes Interface auswählen, das mit einem COM-Objekt, das Sie gerade erstellen, implementiert werden soll. Das ausgewählte Interface wird das Standard-Interface des neu erstellten COM-Objekts. Der COM-Objekt-Experte fügt der erzeugten Implementierungs-Unit Methodenstrukturimplementierungen für alle Methoden dieses Interfaces hinzu. Sie können anschließend in den Rumpf dieser Methoden den Quelltext für die Implementierung des Interface eintragen.

Warnung: Der Experte für Interface-Auswahl fügt das Interface nicht in die Typbibliothek des Projekts ein. Das bedeutet für das Deployment Ihres Objekts, dass Sie auch die Typbibliothek weitergeben müssen, die das Interface Ihres Objekts definiert.

Element	Beschreibung
Interface-Liste	Der Experte führt alle Interfaces auf, die in den registrierten Typbibliotheken definiert sind. Sie können jedes beliebige duale oder benutzerdefinierte Interface aus der Liste auswählen. Für alle Interfaces ist auch der Name der Datei, die die Typbibliothek enthält aufgeführt. Dabei kann es sich um ausführbare Dateien (.exe), Bibliotheken (.dll oder .ocx) oder um Typbibliotheken (.tlb) handeln.
Bibliothek hinzufügen	Wenn sich das gewünschte Interface in einer Typbibliothek befindet, die aktuell nicht registriert ist, klicken Sie auf die Schaltfläche Bibliothek hinzufügen, navigieren zu der Typbibliothek, die das Interface enthält und klicken auf OK. Dadurch wird die ausgewählte Typbibliothek registriert und die Interface-Liste im Experten für Interface-Auswahl aktualisiert.

2.103 Neue ASP.NET-Anwendung

Datei ▶ Neu ▶ Weitere ▶ ASP.NET-Anwendung

Verwenden Sie dieses Dialogfeld zum Festlegen von Optionen für neue Web Forms-Anwendungen.

Element	Beschreibung
Name	Legen Sie den Namen Ihrer ASP.NET-Anwendung fest.
Speicherort	Geben Sie das Stammverzeichnis für Ihre Anwendung ein.
Server	Legen Sie den Standard-Webserver für die neue Webanwendung fest.
Serveroptionen einblenden	Blenden Sie Serveroptionen, wie z.B. Zugriffsrechte, ein oder aus.
Alias	Legen Sie den Namen für den Zugriff auf das virtuelle Verzeichnis fest.
Berechtigungen	Legen Sie die Zugriffsberechtigungen für Ihre Anwendung fest. Diese Optionen beziehen sich nur auf den Microsoft IIS Server. Optionen für den Cassini Web Server können Sie aus der Dropdown-Liste auswählen. Lesen: Ermöglicht lesenden Zugriff auf die Dateien des Verzeichnisses. Scripts (wie z.B. ASP) ausführen: Ermöglicht das Lesen von Webserver-Scripts. Ausführen (wie z.B. ISAPI-Anwendung oder CGI): Ermöglicht die Ausführung von ISAPI- und CGI-Anwendungen. Schreiben: Ermöglicht das Speichern von Dateien in dem Verzeichnis. Durchsuchen: Ermöglicht den Zugriff auf Datei- und Verzeichnislisten.

2.104 Neue Konsolenanwendung

Datei ▶ **Neu** ▶ **Weitere...**

Verwenden Sie dieses Dialogfeld, um eine Anwendung zu erstellen, die in einem Konsolenfenster ausgeführt wird.

Element	Beschreibung
Quelltyp	Legt die Sprache fest, die für das Hauptmodul der Anwendung verwendet wird.
VCL verwenden	Erstellt eine DLL, die VCL-Komponenten enthalten kann. Diese Option steht nur zur Verfügung, wenn Sie C++ als Quelltyp ausgewählt haben. Ist diese Option aktiviert, bezieht die IDE <code>vcl.h</code> ein und ändert den Start-Code und die Linker-Optionen, damit eine Kompatibilität mit den VCL-Objekten hergestellt wird.
Multi-Threaded	Legt mehr als einen Ausführungs-Thread fest. Diese Option ist erforderlich, wenn Sie die Option VCL verwenden aktivieren.
Konsolen-Anwendung	Erstellt ein Konsolenfenster für Ihre Anwendung.
Angabe Projektquelle	der Ermöglicht die Angabe einer vorhandenen Quelldatei für die Konsolenanwendung. Um eine Quelldatei anzugeben, aktivieren Sie diese Option und klicken auf [...], um die Datei auszuwählen.

2.105 Experte für neues DBWeb Control

Datei▶**Neu**▶**Weitere**▶**Delphi für .NET-Projekte**▶**DBWeb Control-Bibliothek**

Verwenden Sie dieses Dialogfeld zum Erstellen eines datensensitiven Web Control. Dieses DBWeb Control kann die DBWeb Controls in der Tool-Palette ergänzen.

Element	Beschreibung
Name des Control	Geben Sie den Namen der Unit an, der das DBWeb Control zugewiesen werden soll.
An DataTable binden	Markieren Sie diese Option, um ein DBWeb Control zu erzeugen, das das IDBWebDataLink-Interface implementiert. Über dieses Interface können Sie auf die Eigenschaften DBDataSource und TableName aller Controls zugreifen.
An DataColumn binden	Markieren Sie diese Option, um ein DBWeb Control zu erzeugen, das das IDBWebColumnLink-Interface implementiert. Über dieses Interface können Sie auf die Eigenschaft ColumnName der Controls zugreifen, um Daten aus einer bestimmten Spalte zu ermitteln.
Lookup unterstützen	Markieren Sie diese Option, um ein DBWeb Control zu erzeugen, das das IDBWebLookupColumnLink-Interface implementiert. Über dieses Interface können Sie auf die Eigenschaften LookupTableName, DataTextField und DataValueField für Lookup-Controls zugreifen. Dieses Kontrollkästchen ist nur aktiviert, wenn die Option An DataColumn binden ausgewählt ist.

2.106 Neue Dynamische Link-Bibliothek

Datei ▶ **Neu** ▶ **Weitere...**

Verwenden Sie dieses Dialogfeld zum Erstellen neuer DLL-Projekte. Eine dynamische Link-Bibliothek ist ein Modul aus kompiliertem Code, das eine Funktionalität für Anwendungen bereit stellt.

Element	Beschreibung
Quelltyp	Legt die Sprache fest, die für das Hauptmodul der DLL verwendet wird.
VCL verwenden	Erstellt eine Anwendung, die VCL-Komponenten enthalten kann. Diese Option steht nur zur Verfügung, wenn Sie C++ als Quelltyp ausgewählt haben. Ist diese Option aktiviert, bezieht die IDE <code>vcl.h</code> ein und ändert den Start-Code und die Linker-Optionen, damit eine Kompatibilität mit den VCL-Objekten hergestellt wird.
Multi-Threaded	Legt mehr als einen Ausführungs-Thread fest. Diese Option ist erforderlich, wenn Sie die Option VCL verwenden aktivieren.
DLL im VC++-Stil	Setzt den DLL-Eintrittspunkt in <code>DLLMain</code> . Lassen Sie diese Option deaktiviert, wenn Sie <code>DLLEntryPoint</code> als Eintrittspunkt verwenden möchten.

2.107 Objektgalerie

Datei ▶ **Neu** ▶ **Weitere...**

Verwenden Sie dieses Dialogfeld zur Erstellung eines neuen Projekts oder einer anderen Entität. Im Dialogfeld Objektgalerie werden Projekt-Templates angezeigt, die in der Objektablage von RAD Studio gespeichert sind.

Element	Beschreibung
Elementkategorien	Klicken Sie auf einen Ordner im Bereich Elementkategorien, um die Typen der Elemente anzuzeigen, die erstellt werden können.

Tip: Wenn Sie mit der rechten Maustaste im rechten Bereich des Fensters klicken, wird ein Kontextmenü mit Befehlen geöffnet, mit denen Sie das Erscheinungsbild dieses Dialogfelds steuern können.

Siehe auch

Templates in die Objektablage einfügen (↗ siehe Seite 58)

2.108 Neue Anwendung

Datei ▶ Neu

Verwenden Sie dieses Dialogfeld zum Festlegen eines Namens und des Speicherorts für die neue Anwendung.

Element	Beschreibung
Name	Geben Sie einen Namen für die Anwendung ein oder akzeptieren Sie den vorgeschlagenen Anwendungsnamen.
Speicherort	Geben Sie einen Verzeichnispfad ein oder navigieren Sie mithilfe der Ellipsen-Schaltfläche zu einem Verzeichnis. Der Standardpfad für die Anwendung wird angezeigt.

2.109 Neues Remote-Datenmodulobjekt

Datei ▶ **Neu** ▶ **Weitere...**

Mit diesem Dialogfeld erstellen Sie ein Datenmodul, das als Automatisierungsserver mit dualityer Schnittstelle für externe Rechner zugänglich ist.

Element	Beschreibung
Name der CoClass	<p>Gibt den Basisnamen für die Automatisierungsschnittstelle des externen Datenmoduls an. Der Klassenname für das Remote-Datenmodul wird der Name der CoClass sein, dem ein T vorangestellt wird. Die Implementierungsklasse erbt von einem Interface mit dem Namen der CoClass, dem ein I vorangestellt ist.</p> <p>Um Client-Anwendungen den Zugriff auf diese Schnittstelle zu ermöglichen, weisen Sie der Eigenschaft ServerName der Verbindungskomponenten der Client-Anwendung den hier angegebenen Namen der CoClass zu.</p>
Threading-Modell	Gbit an, wie Client-Aufrufe an das Interface des Remote-Data-Modul weitergegeben werden.
Modell	Beschreibung
Single	Das Datenmodul bedient zu einem gegebenen Zeitpunkt jeweils nur eine Client-Anforderung. Da die Client-Anforderungen von COM serialisiert werden, müssen Thread-Konflikte nicht addressiert werden.
Apartment	<p>Jede Instanz des externen Datenmoduls erhält zu einem gegebenen Zeitpunkt nur jeweils eine Client-Anforderung. Die DLL kann jedoch mehrere Anwendungen in getrennten Threads bearbeiten, wenn sie mehrere COM-Objekte erstellt. Instanzdaten sind sicher; Sie müssen jedoch sicherstellen, dass im globalen Speicher keine Thread-Konflikte auftreten können.</p> <p>Dies ist das empfohlene Threading-Modell für Datenmengen, die die BDE unterstützen.</p>
Frei	<p>Die Instanzen des externen Datenmoduls können in verschiedenen Threads gleichzeitig mehrere Client-Anforderungen empfangen. Sie müssen sicherstellen, dass die Instanzdaten und der globale Speicher gegen Thread-Konflikte gesichert sind.</p> <p>Dies ist das empfohlene Threading-Modell für ADO-Datenmengen.</p>
Both	Dieses Threading-Modell entspricht im Wesentlichen dem Threading-Modell Frei, die Callback-Routinen für Client-Interfaces sind jedoch serialisiert.
Neutral	<p>Mehrere Clients können das externe Datenmodul gleichzeitig in verschiedenen Threads aufrufen, aber COM stellt sicher, dass die Aufrufe nicht miteinander in Konflikt geraten. Sie müssen globale Daten und Instanzdaten, auf die mehrere Schnittstellenmethoden zugreifen, vor Thread-Konflikten schützen.</p> <p>Das Modell steht nur unter COM+ zur Verfügung. In anderen Umgebungen wird für diese Option das Apartment-Modell eingesetzt.</p>
Beschreibung	Gibt den Text an, der in der Registry neben der ProgID für das Anwendungs-Server-Interface erscheint. Der Beschreibungstext kann auch als Hilfetext für das Interface in der Typbibliothek dienen.
Ereignisunterstützung generieren	Implementiert ein separates Interface zur Verwaltung von Ereignissen.

Siehe auch

[Ereignisse in Ihrem Automatisierungsobjekt verwalten](#)

2.110 Neues Thread-Objekt

Datei ▶ **Neu** ▶ **Weitere** ▶ **Delphi-Projekte** ▶ **Delphi-Dateien** ▶ **Thread-Objekt**

Verwenden Sie dieses Dialogfeld zur Definition einer Thread-Klasse, die einen einzelnen Ausführungs-Thread in einer Multi-Thread-Anwendung kapselt.

Element	Beschreibung
Klassenname	Geben Sie den vollständigen Namen der Klasse ein, die Sie definieren möchten. Dieses Dialogfeld stellt dem Klassennamen kein T voran; geben Sie daher den vollständigen Klassennamen ein (z.B. TMyThread anstatt MyThread).
Benannter Thread	Wenn Sie den Thread benennen möchten, markieren Sie das Kontrollkästchen Benannter Thread und geben im Feld Thread-Name einen Namen ein. Durch die Benennung der Thread-Klasse wird eine Methode namens <i>SetName</i> in die Thread-Klasse eingefügt. Sobald der Thread ausgeführt wird, erfolgt zuerst der Aufruf der Methode <i>SetName</i> . Benannte Threads können im Debugger-Fenster Thread-Status leichter identifiziert werden.
Thread-Name	Geben Sie den Thread-Namen ein, den Sie verwenden möchten.

Tip: Klicken Sie OK an, um eine neue Unit anzulegen, die eine Thread-Klasse mit dem angegebenen Namen definiert. Erstellen Sie anschließend in der neuen Unit den Quelltext für die Methode *Execute*. Dieser Quelltext wird abgearbeitet, wenn der Thread ausgeführt wird.

2.111 Öffnen

Datei Öffnen

Verwenden Sie dieses Dialogfeld zum Suchen und Öffnen einer Datei. Der Titel des Dialogfelds richtet sich nach der auszuführenden Aktion.

Element	Beschreibung
Suchen in	Zeigt den Inhalt des aktuellen Verzeichnisses an. Verwenden Sie die Dropdown-Liste, um ein anderes Laufwerk oder Verzeichnis auszuwählen.
Zum zuletzt besuchten Ordner wechseln	Wechselt in das Verzeichnis, das zuletzt verwendet wurde.
Ein Verzeichnis nach oben	Wechselt zum übergeordneten Verzeichnis.
Neuen Ordner erstellen	Erstellt im aktuellen Verzeichnis ein neues Unterverzeichnis.
Menü Ansicht	Zeigt eine Liste der Dateien und Verzeichnisse zusammen mit Datums-/Zeitangaben, Größe und Attributinformationen an. Für die Anzeige stehen fünf verschiedene Formate zur Verfügung: große Symbole, kleine Symbole, Liste, Details (mit Datums-/Zeitangaben, Größe und Attributinformationen) und Miniaturen (eine verkleinerte Version der Dateigrafik).
Dateien	Zeigt die Dateien im aktuellen Verzeichnis an, die mit dem Suchmuster im Feld Dateiname oder dem Dateityp im Feld Dateityp übereinstimmen. Sie können eine Dateiliste (Standard) oder jede Datei mit allen Einzelheiten anzeigen.
Dateiname	Der Name der Datei, die Sie laden möchten. Sie können Platzhalter eingeben, um die Anzeige im Listenfeld Dateien zu filtern, oder auf den Abwärtspfeil klicken und eine Datei auswählen, die bereits geöffnet war.
Dateityp	Der Typ der zu öffnenden Datei. Im Listenfeld Dateien werden alle Dateien des aktuellen Verzeichnisses angezeigt, die dem gewählten Typ entsprechen.
Öffnen	Öffnet die ausgewählte Datei.

Tip: Drücken Sie in einem Listenfeld oder einer Spalte die Taste F1, um Kurzhinweise mit weiteren Informationen anzuzeigen.

2.112 Package

Datei▶**Neu**▶**Weitere**▶**Delphi für .NET-Projekte**▶**Package**

Verwenden Sie dieses Dialogfeld zum Erstellen neuer Packages.

Element	Beschreibung
Terminologie	Compiliert das aktuelle Package. Wenn Änderungen erforderlich sind, werden diese vor dem Compilieren in einem Dialogfeld angezeigt.
Hinzufügen	Fügt dem Package ein Element hinzu.
Entfernen	Entfernt das markierte Element aus dem Package.
Optionen	Zeigt das Dialogfeld Projektoptionen an.
Installieren	Installiert das aktuelle Package als Entwurfszeit-Package. Wenn Änderungen erforderlich sind, werden diese vor dem Compilieren in einem Dialogfeld angezeigt.
Contains	Zeigt die in das Package eingebundenen Units an. Um eine weitere Unit hinzufügen, klicken Sie auf Hinzufügen. Wenn Sie den Quelltext einer Unit bearbeiten möchten, doppelklicken Sie auf den entsprechenden Eintrag.
Erfordert	Zeigt die anderen Packages an, die für das aktuelle Package erforderlich sind. Um ein Package hinzuzufügen, klicken Sie auf Hinzufügen. Wenn Sie ein Package in einem eigenen Package-Editor anzeigen möchten, doppelklicken Sie auf das Package.

Tip: Wenn Sie im Package-Editor mit der rechten Maustaste klicken, wird ein Kontextmenü mit weiteren Befehlen eingeblendet.

2.113 Druckauswahl

Datei ▶ Drucken

Verwenden Sie dieses Dialogfeld zum Drucken der aktuellen Datei.

Element	Beschreibung
Markierten Block drucken	Bewirkt, dass nur der markierte Textblock gedruckt wird.
Überschrift/Seitennummern	Bewirkt, dass auf jeder Seite eine Überschrift mit dem Dateipfad und dem Dateinamen sowie eine Seitennummer gedruckt wird.
Zeilennummern	Bewirkt die Ausgabe von Zeilennummern auf den gedruckten Seiten.
Syntaxhervorhebung	Bewirkt das Drucken von Syntaxhervorhebungen, z.B. von Schlüsselwörtern in Fettschrift.
Farben verwenden	Bewirkt die farbige Ausgabe von Syntaxhervorhebungen (bei Verwendung eines Farbdruckers).
Zeilen umbrechen	Bewirkt den Umbruch von Zeilen, die breiter als die Druckseite sind.
Linker Rand	Legt den linken Rand der Druckseiten fest.
SetUp	Öffnet das Dialogfeld Druckereinrichtung.

2.114 Projekt-Upgrade

Datei ▶ **Öffnen**

Verwenden Sie dieses Dialogfeld, um für ein älteres Delphi-Projekt, für das keine entsprechende `.bdsproj`-Projektdatei vorliegt, ein Upgrade durchzuführen. Beim Upgrade eines Projekts werden die `.bdsproj`-Datei und andere Dateien und Verzeichnisse, die RAD Studio verwendet, im Projektverzeichnis erstellt.

2.115 Projekte aktualisieren

2

Datei > Öffnen

Dieses Dialogfeld erscheint, wenn ein Projekt aus einer vorherigen Version automatisch für RAD Studio aktualisiert wird. An dem Projekt werden die folgenden Änderungen vorgenommen:

- Die Codierung der Projektdaten wird aktualisiert.
- Referenzen auf [LIB](#)-, [BPI](#)- und [CSM](#)-Dateien werden bei Bedarf aktualisiert.

Anmerkung: In den Versionshinweisen im Hauptordner der Installation finden Sie Informationen zum Thema Kompatibilität.

2.116 Experte für externes Datenmodul

Datei ▶ **Neu** ▶ **Weitere** ▶ **Externes Datenmodul**

Mit dem Experten für externes Datenmodul erstellen Sie ein Datenmodul, das als Automatisierungsserver mit dualem Interface für externe Rechner zugänglich ist. Ein externes Datenmodul befindet sich in einer mehrschichtigen Datenbankumgebung im Anwendungsserver zwischen einem Client und einem Server.

Element	Beschreibung
Name der CoClass	Geben Sie den Basisnamen für das Automatisierungs-Interface des externen Datenmoduls ein. Als Klassennamen für Ihr externes Datenmodul (ein Nachkomme von TRemoteDataModule) wird dieser Name mit einem vorangestellten T verwendet. Die Klasse implementiert ein Interface mit dem Namen der Basisklasse und einem vorangestellten I. Um Client-Anwendungen den Zugriff auf dieses Interface zu ermöglichen, weisen Sie der Eigenschaft ServerName der Verbindungskomponenten der Client-Anwendung den hier angegebenen Namen der Basisklasse zu.
Instantiierung	Wählen Sie aus diesem Kombinationsfeld den Aufrufmodus für Ihre externe Datenmodulanwendung aus. Die folgende Tabelle enthält die möglichen Werte:
Wert	Bedeutung
Intern	Das externe Datenmodul wird in einem In-Process-Server erstellt. Wählen Sie diese Option, wenn ein externes Datenmodul als Teil einer aktiven Bibliothek (DLL) erstellt werden soll.
Eine Instanz	Für jede ausführbare Datei wird nur eine einzige Instanz des externen Datenmoduls erstellt. Jede Client-Verbindung ruft ihre eigene Instanz der ausführbaren Datei auf. Die Instanz des externen Datenmoduls ist deshalb ausschließlich einem einzelnen Client zugeordnet.
Mehrere Instanzen	Eine einzelne Instanz der Anwendung (Prozess) instantiiert alle externen Datenmodule, die für Clients erstellt wurden. Zwar wird jedes externe Datenmodul einer einzigen Client-Verbindung zugeordnet, aber alle Module nutzen denselben Prozessraum.
Threading-Modell	Wählen Sie aus diesem Kombinationsfeld aus, wie Client-Aufrufe an das Interface Ihres externen Datenmoduls übergeben werden sollen. Die folgende Tabelle enthält die möglichen Werte:
Wert	Bedeutung
Einfach	Das Datenmodul empfängt zu einem gegebenen Zeitpunkt jeweils nur eine Client-Anforderung. Weil alle Client-Anforderungen von COM serialisiert werden, müssen Sie sich um Threading-Fragen nicht kümmern.
Apartment	Jede Instanz Ihres externen Datenmoduls erhält zu einem gegebenen Zeitpunkt nur jeweils eine Client-Anforderung. Die DLL kann jedoch mehrere Anwendungen in getrennten Threads bearbeiten, wenn sie mehrere COM-Objekte erstellt. Instanzdaten sind sicher, aber Sie müssen Thread-Konflikte im globalen Speicher berücksichtigen. Bei Verwendung von BDE-konformen Datenmengen sollte immer das Apartment-Modell eingesetzt werden. (Bitte beachten Sie, dass Sie bei der Verwendung von BDE-konformen Datenmengen eine Sitzungskomponente mit der Einstellung "AutoSessionName = True" hinzufügen müssen.)
Frei	Die Instanzen Ihres externen Datenmoduls können in verschiedenen Server-Threads gleichzeitig mehrere Client-Anforderungen empfangen. Sie müssen sowohl die Instanzdaten als auch den globalen Speicher vor Konflikten schützen. Bei Verwendung von ADO-Datenmengen sollte immer dieses Modell eingesetzt werden.
Beides	Entspricht dem Modell "Frei" außer dass alle Callback-Funktionen zu Interfaces serialisiert werden.

Neutral	Mehrere Clients können das externe Datenmodul gleichzeitig in verschiedenen Threads aufrufen, aber COM stellt sicher, dass die Aufrufe nicht miteinander in Konflikt geraten. Sie müssen globale Daten und Instanzdaten, auf die mehrere Interface-Methoden zugreifen, vor Thread-Konflikten schützen. Das Modell steht nur unter COM+ zur Verfügung. In anderen Umgebungen wird für diese Option das Apartment-Modell eingesetzt.
Beschreibung	Geben Sie den Text ein, der in der Registrierung neben der ProgID für das Anwendungsserver-Interface erscheint. Der Beschreibungstext kann auch als Hilfetext für das Interface in der Typbibliothek dienen.
Ereignisunterstützung generieren	Mithilfe dieser Option kann ein eigenes Interface für die Verwaltung der Ereignisse implementiert werden.

2.117 Satellite-Assemblierungsexperte

[Datei](#) ▶ [Neu](#) ▶ [Weitere](#) ▶ [Delphi für .NET-Projekte](#) ▶ [Satellite-Assemblierungsexperte](#)

Verwenden Sie diesen Experten zum Hinzufügen von Satellite-Assemblierungen zu Ihrem Projekt. Folgen Sie den Anweisungen auf den einzelnen Registerseiten des Experten.

Siehe auch

Übersetzungs-Tools verwenden ( siehe Seite 1437)

Sprachen zu einem Projekt hinzufügen ( siehe Seite 86)

2.118 Speichern unter

Datei▶Speichern unter

Verwenden Sie dieses Dialogfeld zum Speichern der aktiven Datei.

Element	Beschreibung
Speichern in	Zeigt den Inhalt des aktuellen Verzeichnisses an. Verwenden Sie die Dropdown-Liste, um ein anderes Laufwerk oder Verzeichnis auszuwählen.
Zum zuletzt besuchten Ordner wechseln	Wechselt in das Verzeichnis, das zuletzt verwendet wurde.
Ein Verzeichnis nach oben	Wechselt zum übergeordneten Verzeichnis.
Neuen Ordner erstellen	Erstellt im aktuellen Verzeichnis ein neues Unterverzeichnis.
Menü Ansicht	Zeigt eine Liste der Dateien und Verzeichnisse zusammen mit Datums-/Zeitangaben, Größe und Attributinformationen an. Für die Anzeige stehen fünf verschiedene Formate zur Verfügung: große Symbole, kleine Symbole, Liste, Details (mit Datums-/Zeitangaben, Größe und Attributinformationen) und Miniaturen (eine verkleinerte Version der Dateigrafik).
Dateien	Zeigt die Dateien im aktuellen Verzeichnis an, die mit dem Suchmuster im Feld Dateiname oder dem Dateityp im Feld Dateityp übereinstimmen. Sie können eine Dateiliste (Standard) oder jede Datei mit allen Einzelheiten anzeigen.
Dateiname	Der Name der Datei, die Sie speichern möchten. Sie können Platzhalter eingeben, um die Anzeige im Listenfeld Dateien zu filtern, oder auf den Abwärtspfeil klicken und eine Datei auswählen, die bereits gespeichert wurde.
Dateityp	Der Typ der zu speichernden Datei. Im Listenfeld Dateien werden alle Dateien des aktuellen Verzeichnisses angezeigt, die dem gewählten Typ entsprechen.
Speichern	Speichert die ausgewählte Datei.

Tip: Drücken Sie in einem Listenfeld oder einer Spalte die Taste F1, um Kurzhinweise mit weiteren Informationen anzuzeigen.

2.119 Verzeichnis auswählen

Verwenden Sie dieses Dialogfeld, um ein Arbeitsverzeichnis für Ihr neues Projekt festzulegen.

Element	Beschreibung
Verzeichnisname	Zeigt das aktuelle Verzeichnis an. Wenn Sie ein noch nicht vorhandenes Verzeichnis eingeben, wird dieses automatisch erstellt.
Speichern in	Zeigt den Inhalt des aktuellen Verzeichnisses an.
Dateien: (*.*)	Listet alle Dateien in dem aktuellen Verzeichnis auf. Sie können von diesen Dateien keine auswählen. Die Liste wird nur angezeigt, damit Sie den Inhalt des aktuellen Verzeichnisses kennen.
Laufwerke	Listet alle verfügbaren Laufwerke auf. Sie können eines dieser Laufwerke auswählen.

2.120 Neuen Web-Service hinzufügen

Datei ▶ **Neu** ▶ **Weitere** ▶ **Delphi-Projekte** ▶ **WebServices** ▶ **SOAP-Server-Interface**

Verwenden Sie dieses Dialogfeld zur Definition eines neuen aufrufbaren Interface und dessen Implementierungsklasse. Das Dialogfeld erzeugt eine neue Unit und deklariert ein aufrufbares Interface sowie die Implementierungsklasse. Das Interface ist von *IInvokable* und der Implementierungsklasse von *TInvokableClass* abgeleitet. Generiert wird auch der Quelltext zur Registrierung des Interface und der Implementierungsklasse in der Aufrufregistrierung. Nach Beendigung des Experten können Sie das generierte Interface und die Klassendefinitionen bearbeiten, indem Sie die Eigenschaften und Methoden hinzufügen, die als Web-Service zur Verfügung stehen sollen.

Element	Beschreibung
Service-Name	Geben Sie einen Namen für das aufrufbare Interface (Port-Typ) ein, das Ihr Web-Service für Clients zur Verfügung stellt. Der Name wird dem Interface zugewiesen. Dieser Name wird auch zur Generierung des Namens der Implementierungsklasse benutzt. Wenn Sie z.B. <i>MyWebService</i> eingeben, generiert der Experte die Definition eines aufrufbaren Interface mit dem Namen <i>MyWebService</i> sowie eine Implementierungsklasse mit der Bezeichnung <i>TMyWebServiceImpl</i> .
Unit-Bezeichner	Geben Sie einen Namen für die Unit ein, die der Experte erstellen soll. Diese Unit enthält die Definitionen des Interface und der Implementierungsklasse.
Kommentare generieren	Optional. Fügt der Unit vom Experten erzeugte Kommentare hinzu, die beschreiben, was der Code ausführt.
Beispiel-Methoden generieren	Optional. Fügt der Unit vom Experten erzeugten Beispielcode als Kommentar hinzu. Sie können diesen Beispielcode als Richtlinie zum Definieren und Implementieren des aufrufbaren Interface und der Implementierungsklasse nutzen.
Service-Aktivierungsmodell	Wählen Sie das gewünschte Aktivierungsmodell aus der Drop-Down-Liste: Auf Anforderung erstellt eine neue Instanz der Implementierungsklasse als Reaktion auf empfangene Anforderungen. Nach Behandlung der Anforderung wird die Instanz freigegeben. Global erstellt nur eine Instanz der Implementierungsklasse, die sämtliche Anforderungen behandelt.

2.121 SOAP-Datenmodul-Experte

Datei▶**Neu**▶**Weitere**▶**Delphi-Projekte**▶**WebServices**▶**SOAP-Server-Datenmodul**

Verwenden Sie dieses Dialogfeld zum Hinzufügen eines SOAP-Datenmoduls zu einer Web-Service-Anwendung. Mit SOAP-Datenmodulen können Web-Service-Anwendungen Datenbankinformationen als Web-Service exportieren. Client-Datenmengen in der Client-Anwendung können diese Datenbankinformationen anzeigen und aktualisieren.

Element	Beschreibung
Modulname	Geben Sie den Basisnamen der von <i>TSoapDataModule</i> abgeleiteten Klasse ein, die Ihre Anwendung erzeugt. Dieser Name dient auch als Basisname für das Interface für diese Klasse. Wenn Sie beispielsweise den Klassennamen <i>MyDataServer</i> angeben, erzeugt der Experte eine neue Unit, in der <i>TMyDataServer</i> (ein Nachkomme von <i>TSoapDataModule</i>) deklariert wird, der <i>IMyDataServer</i> (einen Nachkomme von <i>IAppServerSOAP</i>) implementiert.

2.122 Neue SOAP-Server-Anwendung

Datei ▶ **Neu** ▶ **Weitere** ▶ **Delphi-Projekte** ▶ **WebServices** ▶ **SOAP-Server-Anwendung**

Verwenden Sie dieses Dialogfeld, um den Server-Typ festzulegen, mit dem Ihre Web-Service-Anwendung arbeiten soll.

Element	Beschreibung
ISAPI/NSAPI-DLL	ISAPI- und NSAPI-Server-Anwendungen sind DLLs, die vom Web-Server geladen werden. Client-Anforderungsinformationen werden an die DLL als Struktur übergeben. Jede Anforderung wird in einem eigenen Thread behandelt.
CGI, ausführbare Datei	Eine ausführbare CGI-Datei ist eine Konsolenanwendung, die Client-Anforderungen von der Standardeingabe entgegennimmt, sie verarbeitet und das Ergebnis über die Standardausgabe an den Server zurückgibt. Jede Anforderung wird in einer eigenen Instanz der Anwendung behandelt.
Web-Debugger-Anwendung	Der Webanwendungs-Debugger stellt eine einfache Möglichkeit zur Verfolgung, HTTP-Anforderungen, -Antworten und -Antwortzeiten zu überwachen. Der Webanwendungs-Debugger tritt an die Stelle des Webservers. Nach dem Debuggen Ihrer Anwendung können Sie sie in einen anderen Web-Anwendungstyp konvertieren und sie mit einem kommerziellen Web-Server installieren.
Klassenname	Wenn Web-Debugger-Anwendung ausgewählt ist, müssen Sie einen Klassennamen für den Debugger angeben, über den das Web-Modul aufgerufen werden kann.

2.123 Transaktionaler Objektexperte

[Datei](#) ▶ [Neu](#) ▶ [Weitere...](#)

Verwenden Sie den Transaktionalen Objektexperten zum Erstellen eines Server-Objekts, das unter MTS oder COM+ ausführbar ist. Transaktionale Objekte werden in verteilten Anwendungen eingesetzt, um die speziellen Dienste von MTS oder COM+ für die Ressourcenverwaltung, die Transaktionsunterstützung oder die Sicherheit nutzen zu können.

Element	Beschreibung
Name der CoClass	Geben Sie den Namen des Objekts ein, das implementiert werden soll. Der Experte erzeugt ein Interface mit diesem Namen und einem vorangestellten 'I' sowie eine Implementierungsklasse mit diesem Namen und einem vorangestellten 'T'.
Threading-Modell	Wählen Sie hier das Threading-Modell aus, um festzulegen, wie MTS oder COM die Aufrufe des Interface des transaktionalen Objekts serialisiert. Das gewählte Threading-Modell bestimmt, wie das Objekt registriert wird. Sie müssen sicherstellen, dass die Objektimplementierung dem gewählten Modell entspricht. Die Werte für das Threading-Modell werden weiter unten beschrieben.
Transaktionsmodell	Legt das Transaktionsattribut fest, das bei der Registrierung Ihrem Objekt zugewiesen wird. Die möglichen Werte werden weiter unten beschrieben.
Ereignisunterstützung generieren	

Das Kombinationsfeld Threading-Modell kann die folgenden Werte enthalten:

Modell	Beschreibung
Einfach	Der Quelltext verfügt über keine Thread-Unterstützung. Es kann immer nur ein Client-Thread behandelt werden.
Apartment	Unter COM+ greift auf jedes von einem Client instantiierte Objekt immer nur ein Thread zu. Sie müssen Schutzmaßnahmen implementieren, die verhindern, dass mehrere Threads auf den globalen Speicher zugreifen. Objekte können aber sicher auf ihre eigenen Instanzdaten (Eigenschaften und Elemente) zugreifen. Unter MTS verwenden alle Client-Aufrufe den Thread, in dem das Objekt erstellt wurde.
Beides	Wie Apartment, mit der Ausnahme, dass Callbacks an Clients auch serialisiert werden.
Neutral	Mehrere Clients können das Objekt gleichzeitig in verschiedenen Threads aufrufen, aber COM stellt sicher, dass die Aufrufe nicht miteinander in Konflikt geraten. Sie müssen Thread-Konflikte verhindern, die globale Daten und Instanzdaten betreffen, auf die von mehreren Methoden zugegriffen wird. Dieses Modell sollte nicht bei Objekten mit Benutzeroberflächen verwendet werden. Das Modell steht nur unter COM+ zur Verfügung. Es wird unter COM auf das Apartment-Modell abgebildet.

Anmerkung: Die Serialisierung der Objektaufrufe hängt auch von der Art und Weise ab, in der das Objekt an Aktivitäten beteiligt ist. Unter MTS werden Objekte immer durch die aktuelle Aktivität synchronisiert. Die entsprechenden Einstellungen können in der Registerkarte COM+ des Typbibliothekseditors oder mit dem Komponentenmanager von COM+ vorgenommen werden.

Das Kombinationsfeld Transaktionsmodell kann die folgenden Werte enthalten:

Wert	Bedeutung
Transaktion erforderlich	Das Objekt muss in einer Transaktion ausgeführt werden. Wenn ein neues Objekt erstellt wird, erbt sein Objektkontext die Transaktion vom Kontext des Client. Verfügt der Client nicht über einen Transaktionskontext, wird automatisch ein neuer erzeugt.
Neue Transaktion erforderlich	Das Objekt muss in einer eigenen Transaktion ausgeführt werden. Beim Erstellen eines neuen Objekts wird automatisch auch ein neuer Transaktionskontext erstellt. Dies erfolgt unabhängig davon, ob der Client über eine Transaktion verfügt. Das Objekt kann niemals in der Transaktion seines Clients ausgeführt werden. Stattdessen werden vom System stets unabhängige Transaktionen für die neuen Objekte erstellt.
Unterstützt Transaktionen	Das Objekt kann in den Transaktionen seines Clients ausgeführt werden. Wenn ein neues Objekt erstellt wird, erbt sein Objektkontext die Transaktion vom Kontext des Client, falls vorhanden. Ansonsten wird das Objekt nicht in einer Transaktion erstellt.
Unterstützt Transaktionen keine	Unter MTS verhält sich diese Einstellung wie Ignoriert Transaktionen unter COM+ (siehe weiter unten). Unter COM+ kann das Objekt in dem Kontext einer Transaktion überhaupt nicht ausgeführt werden. Wenn der Client über eine Transaktion verfügt, schlagen Versuche, das Objekt zu erstellen, fehl.
Ignoriert Transaktionen	Das Objekt wird nicht in einer Transaktion ausgeführt. Wenn ein neues Objekt erstellt wird, wird sein Objektkontext ohne Transaktion erzeugt. Dabei spielt es keine Rolle, ob sein Client eine Transaktion besitzt. Dieses Modell wird unter MTS nicht unterstützt.

2.124 Unit verwenden

Datei ▶ Unit verwenden

Verwenden Sie dieses Dialogfeld, um den Inhalt einer Unit der aktuellen Unit zur Verfügung zu stellen. Dieses Dialogfeld enthält eine Liste aller Units im Projekt, die noch nicht in die aktuelle Unit eingebunden sind. Sie können nur Units des aktuellen Projekts auswählen.

2.125 Neue Web-Server-Anwendung

Datei ▶ **Neu** ▶ **Weitere** ▶ **Delphi-Projekte** ▶ **Neu** ▶ **Web-Server-Anwendung**

Verwenden Sie dieses Dialogfeld, um den Server-Typ festzulegen, mit dem Ihre Web-Server-Anwendung arbeiten soll.

Element	Beschreibung
ISAPI/NSAPI-DLL	ISAPI- und NSAPI-Server-Anwendungen sind gemeinsame Objekte, die vom Web-Server geladen werden. Die Client-Anforderungsinformation wird der DLL als Struktur übergeben und von <i>TISAPIApplication</i> ausgewertet. Jede Anforderung wird in einem eigenen Thread behandelt. Wenn Sie diesen Anwendungstyp in Delphi wählen, werden der Bibliotheks-Header der Projektdateien und die erforderlichen Einträge der uses -Liste und der exports -Klausel der Projektdatei hinzugefügt.
CGI-Einzelanwendung	Eine ausführbare CGI-Datei ist eine Konsolenanwendung, die Client-Anforderungen von der Standardeingabe entgegennimmt, sie verarbeitet und das Ergebnis über die Standardausgabe an den Server zurückgibt. Diese Daten werden von <i>TCGIApplication</i> verarbeitet. Jede Anforderung wird in einer eigenen Instanz der Anwendung behandelt. Wenn Sie diesen Anwendungstyp in Delphi wählen, werden automatisch die entsprechenden Einträge in die uses -Klausel der Projektdatei aufgenommen, und die erforderliche \$APPTYPE-Anweisung wird in den Quelltext eingefügt.
Web-Debugger-Anwendung	Der Webanwendungs-Debugger stellt eine einfache Möglichkeit zur Verfügung, HTTP-Anforderungen, -Antworten und -Antwortzeiten zu überwachen. Der Webanwendungs-Debugger tritt an die Stelle des Webservers. Nach dem Debuggen Ihrer Anwendung können Sie sie in einen anderen Web-Anwendungstyp konvertieren und sie mit einem kommerziellen Web-Server installieren. Bei der Wahl dieser Anwendungsart müssen Sie einen CoClass-Namen für die Debugger-Anwendung angeben. Mit diesem Namen referenziert die Debugger-Anwendung Ihre Anwendung. Die meisten Entwickler benutzen einfach den Anwendungsnamen als CoClass-Namen.

2.126 Neuen Web-Service hinzufügen

Datei ▶ **Neu** ▶ **Weitere** ▶ **Delphi-Projekte** ▶ **WebServices** ▶ **SOAP-Server-Interface**

Verwenden Sie dieses Dialogfeld zum Erzeugen einer neuen Unit, die ein aufrufbares Interface und dessen Implementierungsklasse deklariert. Das Interface ist von *IInvokable* und der Implementierungsklasse von *TInvokableClass* abgeleitet. Generiert wird auch der Quelltext zur Registrierung des Interface und der Implementierungsklasse in der Aufrufregistrierung. Nach Beendigung des Experten können Sie das generierte Interface und die Klassendefinitionen bearbeiten, indem Sie die Eigenschaften und Methoden hinzufügen, die als Web-Service zur Verfügung stehen sollen.

Element	Beschreibung
Service-Name	Geben Sie einen Namen für das aufrufbare Interface (Port-Typ) ein, das Ihr Web-Service für Clients zur Verfügung stellt. Der Name wird dem Interface zugewiesen. Dieser Name wird auch zur Generierung des Namens der Implementierungsklasse benutzt. Wenn Sie z.B. <i>MyWebService</i> eingeben, generiert der Experte die Definition eines aufrufbaren Interface mit dem Namen <i>MyWebService</i> sowie eine Implementierungsklasse mit der Bezeichnung <i>TMyWebServiceImpl</i> .
Unit-Bezeichner	Geben Sie einen Namen für die Unit ein, die der Experte erstellen soll. Diese Unit enthält die Definitionen des Interface und der Implementierungsklasse.
Kommentare erzeugen	Fügt der Unit Kommentare hinzu, die beschreiben, was der Code ausführt.
Beispiel-Methoden generieren	Fügt der Unit Beispielcode als Kommentar hinzu. Sie können diesen Beispielcode als Richtlinie zum Definieren und Implementieren des aufrufbaren Interface und der Implementierungsklasse nutzen.
Service-Aktivierungsmodell	Wählen Sie das gewünschte Aktivierungsmodell aus der Drop-Down-Liste: Auf Anforderung erstellt eine neue Instanz der Implementierungsklasse als Reaktion auf empfangene Anforderungen. Nach Behandlung der Anforderung wird die Instanz freigegeben. Global erstellt nur eine Instanz der Implementierungsklasse, die sämtliche Anforderungen behandelt.

2.127 Seitenoptionen des Anwendungsmoduls/Neues WebSnap-Seitenmodul

Verwenden Sie dieses Dialogfeld zur Definition der grundlegenden Eigenschaften eines Seitenmoduls. Das Dialogfeld wird – je nachdem, wie Sie darauf zugegriffen haben – mit einem anderen Titel angezeigt.

Element	Beschreibung
Typ	Der Generatortyp der Seite kann auf <i>AdapterPageProducer</i> , <i>DataSetPageProducer</i> , <i>InetXPageProducer</i> , <i>PageProducer</i> oder <i>XSLPageProducer</i> gesetzt werden.
Script-Engine	Wenn der gewählte Seitengenerator das Scripting unterstützt, dann wählen Sie aus der Dropdown-Liste Script-Engine die Script-Sprache für die Seite aus. <i>AdapterPageProducer</i> unterstützt nur JScript.
Neue Datei	Erzeugt eine Template-Datei und verwaltet sie als Teil der Unit. Eine verwaltete Template-Datei wird in der Projektverwaltung angezeigt und hat denselben Dateinamen und denselben Speicherort wie die Unit-Quelldatei. Deaktivieren Sie die Option Neue Datei, wenn Sie die Eigenschaften der Generatorkomponente (normalerweise die Eigenschaft <i>HTMLDoc</i> oder <i>HTMLFile</i>) verwenden möchten.
Template	Wenn Neue Datei markiert ist, wählen Sie aus der Dropdown-Liste Template den Standardinhalt für die Template-Datei aus. Die Standard-Template zeigt den Titel der Anwendung, den Titel der Seite sowie die Hyperlinks zu den publizierten Seiten an.
Name	Geben Sie einen Setennamen und -titel für das Seitenmodul ein. Der Seitenname referenziert die Seite in einer HTTP-Anforderung oder in der Anwendungslogik.
Titel	Geben Sie den Namen ein, der dem Endanwender angezeigt wird, wenn er die Seite in einem Browser öffnet.
Publiziert	Markieren Sie Publiziert, damit die Seite automatisch auf HTTP-Anforderungen antworten kann, in denen der Seitenname mit den Pfadinformationen übereinstimmt.
Anmeldung erfolgreich	Erfordert, dass der Anwender sich anmeldet, damit er auf die Seite zugreifen kann.
Erzeugung	Ist nur im Dialogfeld Neues WebSnap-Seitenmodul enthalten. Dieser Parameter steuert, wann eine Instanz dieses Moduls erzeugt wird. Wenn die Instanz nur erzeugt werden soll, wenn sie referenziert wird, wählen Sie Bei Anforderung. Wenn die Instanz beim Start erzeugt werden soll, wählen Sie Immer.
Pufferung	Ist nur im Dialogfeld Neues WebSnap-Seitenmodul enthalten. Dieser Parameter steuert, wann ein Modul freigegeben wird. Zur Laufzeit kann die Instanz dieses Moduls im Puffer verbleiben oder aus dem Arbeitsspeicher freigegeben werden, wenn die Anforderung bedient wurde. Wählen Sie Instanz puffern, um die Instanz im Arbeitsspeicher zu halten, auch wenn aktuell kein anderes Objekt darauf Bezug nimmt. Wählen Sie Instanz freigeben, um das Modul aus dem Arbeitsspeicher zu entfernen, wenn keine Referenzierungen vorliegen.

Siehe auch

Überblick zu WebSnap

Eine WebSnap-Anwendung erstellen

2.128 Neue WebSnap-Anwendung

Datei ▶ **Neu** ▶ **Weitere** ▶ **Delphi-Projekte** ▶ **WebSnap** ▶ **WebSnap-Anwendung**

Verwenden Sie dieses Dialogfeld zum Konfigurieren einer neuen WebSnap-Anwendung.

Element	Beschreibung
ISAPI/NSAPI-DLL	ISAPI- und NSAPI-Server-Anwendungen sind gemeinsame Objekte, die vom Web-Server geladen werden. Die Client-Anforderungsinformation wird der DLL als Struktur übergeben und von <i>TISAPIApplication</i> ausgewertet. Jede Anforderung wird in einem eigenen Thread behandelt. Wenn Sie diesen Anwendungstyp in Delphi wählen, werden der Bibliotheks-Header der Projektdateien und die erforderlichen Einträge der uses -Liste und der exports -Klausel der Projektdatei hinzugefügt.
CGI-Einzelanwendung	Eine ausführbare CGI-Datei ist eine Konsolenanwendung, die Client-Anforderungen von der Standardeingabe entgegennimmt, sie verarbeitet und das Ergebnis über die Standardausgabe an den Server zurückgibt. Diese Daten werden von <i>TCGIApplication</i> verarbeitet. Jede Anforderung wird in einer eigenen Instanz der Anwendung behandelt. Wenn Sie diesen Anwendungstyp in Delphi wählen, werden automatisch die entsprechenden Einträge in die uses -Klausel der Projektdatei aufgenommen, und die erforderliche \$APPTYPE-Anweisung wird in den Quelltext eingefügt.
Web-Debugger-Anwendung	Mit dem Web-Anwendungs-Debugger können Sie HTTP-Anforderungen, -Antworten und -Antwortzeiten überwachen. Die Web-Debugger-Anwendung tritt an die Stelle des Web-Servers. Nach dem Debuggen Ihrer Anwendung können Sie sie in einen anderen Web-Anwendungstyp konvertieren und sie mit einem kommerziellen Web-Server installieren. Bei der Wahl dieser Anwendungsart müssen Sie einen Klassennamen für die Debugger-Anwendung angeben. Mit diesem Namen nimmt die Web-Debugger-Anwendung auf Ihre Anwendung Bezug. Die meisten Entwickler benutzen den Anwendungsnamen als Klassennamen.
Seitenmodul	Zeigt ein Web-Seitenmodul an, das die Komponenten <i>PageProducer</i> , <i>WebAppServices</i> , <i>ApplicationAdapter</i> , <i>LogicalPageDispatcher</i> und <i>AdapterDispatcher</i> enthält. Wenn ein Web-Seitenmodul im Quelltext-Editor geöffnet ist, können Sie die Unit einer Web-Seite, den HTML-Code und, nachdem das Modul kompiliert und ausgeführt wurde, eine Vorschau der Web-Seite anzeigen.
Datenmodul	Zeigt ein Web-Seitenmodul an, das die Komponenten <i>PageProducer</i> , <i>WebAppServices</i> , <i>ApplicationAdapter</i> , <i>LogicalPageDispatcher</i> und <i>AdapterDispatcher</i> enthält.
Komponenten	Zeigt das Dialogfeld Web-Anwendungskomponenten an, in dem Sie Komponenten für Ihre Anwendung auswählen können.
Seitenname	Wenn Sie als Anwendungsmodultyp Seitenmodul gewählt haben, können Sie die Seite benennen, indem Sie einen Namen in dieses Feld eingeben.
Seitenoptionen	Zeigt das Dialogfeld Seitenoptionen des Anwendungsmoduls an, in dem Sie die grundlegenden Eigenschaften des Seitenmoduls definieren können.
Pufferung	Zur Laufzeit kann die Instanz dieses Moduls im Puffer verbleiben oder aus dem Arbeitsspeicher freigegeben werden, wenn die Anforderung bedient wurde. Wählen Sie eine der folgenden Optionen aus: Wählen Sie Instanz puffern, um die Instanz zwischen Benutzersitzungen im Arbeitsspeicher zu halten. Wählen Sie Instanz freigeben, um das Modul aus dem Arbeitsspeicher zu entfernen, wenn die aktive Sitzung beendet ist.

Siehe auch

[Überblick zu WebSnap](#)

[Eine WebSnap-Anwendung erstellen](#)

2.129 Neues WebSnap-Datenmodul

Verwenden Sie dieses Dialogfeld, um festzulegen, wie der Server die Erzeugung und Freigabe des Datenmoduls vornehmen soll.

Element	Beschreibung
Erzeugung	Steuert, wann eine Instanz dieses Moduls erzeugt wird. Bei Anforderung erzeugt die Instanz nur, wenn sie referenziert ist. Immer erzeugt die Instanz beim Start.
Pufferung	Steuert, wann ein Modul freigegeben wird. Zur Laufzeit kann die Instanz dieses Moduls im Puffer verbleiben oder aus dem Arbeitsspeicher freigegeben werden, wenn die Anforderung bedient wurde. Instanz puffern hält die Instanz im Arbeitsspeicher, auch wenn aktuell kein anderes Objekt darauf Bezug nimmt. Instanz freigeben entfernt das Modul aus dem Arbeitsspeicher, wenn keine Referenzierungen vorliegen.

Siehe auch

[Überblick zu WebSnap](#)

[Eine WebSnap-Anwendung erstellen](#)

2.130 Web-Anwendungskomponenten

Verwenden Sie dieses Dialogfeld, um Komponentenkategorien und aus den Kategorien spezielle Komponenten auszuwählen (in manchen Kategorien steht nur eine Wahlmöglichkeit zur Verfügung).

Element	Beschreibung
Anwendungs-Adapter	Enthält die Feld- und Aktionskomponenten, die über die Scriptvariable <i>Application</i> verfügbar sind.
Endanwender-Adapter	Stellt Informationen über einen Benutzer bereit, wie z.B. Name, Zugriffsrechte und Login-Status. <i>TEndUserAdapter</i> ruft zum Ermitteln von Benutzerinformationen Ereignisbehandlungs routinen auf.
Seiten-Dispatcher	Leitet HTTP-Anforderungen weiter, die ein Web-Seitenmodul anhand des Namens referenzieren.
Dispatcher für Adapter	Behandelt HTML-Formulare und Anforderungen von dynamischen Bildern, indem Adapter-Aktions- und -Feldkomponenten aufgerufen werden.
Dispatch-Aktionen	Übergibt eine HTTP-Anforderungsbotschaft an die entsprechenden Aktionselemente, die eine Antwort zusammenstellen.
Dateisuche-Service	Steuert die Suche nach Templates und include-Dateien zur Laufzeit.
Sitzungs-Service	Speichert Informationen über Endanwenderdaten, die für eine kurze Zeitspanne erforderlich sind. Mit <i>TSessionsService</i> können beispielsweise alle Anwender protokolliert werden, die aktuell eingeloggt sind und ein Anwender automatisch ausgeloggt werden, wenn er eine gewisse Zeit inaktiv war.
Benutzerlisten-Service	Enthält eine Liste mit Benutzernamen, Passwörtern und Zugriffsrechten. Anhand dieser Liste werden Logins ausgewertet und Zugriffsrechte eines bestimmten Anwenders überprüft.

Siehe auch

[Überblick zu WebSnap](#)

[Eine WebSnap-Anwendung erstellen](#)

2.131 WSDL: Optionen für den Import

Verwenden Sie dieses Dialogfeld zur Eingabe von Informationen, die das Importprogramm benötigt, um eine Verbindung mit dem Server herzustellen, der das WSDL-Dokument publiziert, oder um zu konfigurieren, wie der Experte Quelltext für die Definitionen in einem WSDL-Dokument generieren soll.

Registerkarte Verbindung

Auf der Registerseite Verbindung geben Sie die Informationen ein, die der Experte benötigt, um eine Verbindung mit dem Server herzustellen, der das WSDL-Dokument bereitstellt.

Element	Beschreibung
Benutzername	Geben Sie den Benutzernamen an, der verwendet werden soll, wenn sich das WSDL-Dokument auf einem sicheren Server befindet, für den eine Authentifizierung erforderlich ist.
Passwort	Geben Sie das Passwort ein, das neben dem Benutzernamen zur Authentifizierung auf dem Server mit dem WSDL-Dokument erforderlich ist.
Proxy	Geben Sie die Host-Namen für jeden Proxy-Server an, der Anforderungen an die auf der Registerseite Quelle des Dialogfeldes Web-Service-Import angegebene URL weiterleiten muss.

Registerkarte Code-Erzeugung

Auf der Registerseite Code-Erzeugung definieren Sie, wie das Importprogramm die Definitionen des WSDL-Dokuments in Quelltext umsetzen soll.

Element	Beschreibung
Namespace deklarieren	Diese Option ist nur wirksam, wenn das Importprogramm C++ Quelltext generiert. Ist sie ausgewählt, platziert das Importprogramm alle erzeugten Typdefinitionen in einen C++ Namensbereich, der den Namen des importierten Service hat.
Ein OutParam ist Return	Ist diese Option ausgewählt, ordnet das Importprogramm Operationen mit einer einzelnen Ausgabenachricht solchen Funktionen zu, deren Ausgabenachricht der Rückgabewert ist. Andernfalls wird die Ausgabenachricht einem Ausgabeparameter zugeordnet.
Literalparameter abwickeln	Bei der Benutzung von Document Literal Encoding beschreibt der Web-Service keine Operationen, sondern zwei Strukturen. Stattdessen beschreibt er zwei Records, wovon einer die erwartete Ausgabe und der andere die erwartete Eingabe beschreibt. Ist die Option Literalparameter abwickeln ausgewählt, konvertiert das Importprogramm diese beiden Strukturen in Methodenaufrufe.
Destruktoren generieren	Ist diese Option ausgewählt, generiert das Importprogramm Destruktoren für Typklassen. Diese Destruktoren geben alle verschachtelten Elemente frei, deren Typen Klassen von Klassen-Arrays sind. Die generierten Destruktoren vereinfachen für Sie die Freigabe der Instanzen von Klassen, die Typen repräsentieren, da Sie Klassenelemente, die ebenfalls Klassen benutzen, um den Typ <i>TRemotable</i> zu repräsentieren, nicht mehr explizit freigeben müssen.
Schema-Fehler ignorieren	Ist diese Option ausgewählt, versucht der Importer auch WSDL-Dokumente zu importieren, die nicht wohlgeformt sind. Der Importer kann die erforderliche Information oftmals auch dann herleiten, wenn das Schema inkorrekt ist.
Warnungskommentare	Ist diese Option ausgewählt, schreibt das Importprogramm Warnungen in die Kommentare, die es am Anfang der generierten Dateien (sowohl .cpp als .hpp auch) einfügt. Diese Warnungen beschreiben Probleme wie ungültige Typdefinitionen in einem WSDL-Dokument, wenn Schema-Fehler ignorieren markiert ist, Fehler, die aufgetreten sind, wenn Literalparameter abwickeln markiert ist, usw.

Literale ausgeben	Typen	Bei der Benutzung von Document Literal Encoding beschreibt der Web-Service keine Operationen, sondern zwei Strukturen. Stattdessen beschreibt er zwei Records, wovon einer die erwartete Ausgabe und der andere die erwartete Eingabe beschreibt. Ist die Option Literale Typen ausgeben ausgewählt, generiert das Importprogramm Typdefinitionen für diese beiden Strukturen, auch wenn Literalparameter abwickeln aktiviert ist.
Zweideutige Typen als Array		Sofern ein WSDL-Dokument Schemadefinitionen nicht konsistent verwendet, hat der Importer Probleme beim importieren von Array-Typen. Dadurch ergibt sich ein Typ, der durch eine Klasse ohne Elemente dargestellt wird. Ist die Option Zweideutige Typen als Array ausgewählt, wandelt das Importprogramm solche Klassen in Arrays um. Diese Option kann problemlos aktiviert werden, solange das WSDL-Dokument keinen Web-Service beschreibt, der Document Literal Encoding benutzt. Document Literal Encoding kann auch in einer leeren Klasse resultieren, die eine Prozedur darstellt. Benutzt ein Web-Service Document Literal Encoding und ist die Option Zweideutige Typen als Array aktiviert, wird der generierte Code möglicherweise nicht funktionieren.
Server-Implementierung generieren		Ist diese Option ausgewählt, generiert das Importprogramm Implementierungsklassen für die importierten Interfaces. Benutzen Sie diese Option, wenn Sie einen Server entwickeln, der einen bereits in einem WSDL-Dokument definierten Web-Service implementiert.
String als WideString		Ist diese Option ausgewählt, wandelt das Importprogramm alle String-Typen in WideString-Werte um. Andernfalls benutzt das Importprogramm den String-Typ. WideString-Werte können für erweiterte Zeichensätze erforderlich sein. Enthalten Strings keine derartigen Zeichen, ist der String-Typ effizienter.

2.132 Experte fÃ¼r den WSDL-Import

Datei►**Neu**►**Weitere**►**Delphi-Projekte**►**WebServices**►**WSDL-Import**

Verwenden Sie dieses Dialogfeld zum Importieren eines WSDL-Dokuments oder eines XML-Schemas, das einen Web-Service beschreibt. Nach dem Import erzeugt der Experte alle Interface- und Klassendefinitionen, die fÃ¼r das Aufrufen dieser Web-Services Ã¼ber ein externes Interface-Objekt (THTTPRIO) erforderlich sind. Der Experte kann auch ein von Ihnen auszufÃ¼llendes CodegerÃ¼st fÃ¼r eine Web-Service-Anwendung erstellen (wenn Sie beispielsweise einen Web-Service implementieren mÃ¶chten, der bereits in einem WSDL-Dokument definiert ist).

Seite Quelle

Auf der Seite Quelle des Experten kÃ¶nnen Sie den Namen des zu importierenden WSDL-Dokuments oder XML-Schemas festlegen.

Element	Beschreibung
Verzeichnis der WSDL-Datei oder URL	<p>Geben entweder einen WSDL-Dateinamen oder die URL der Site ein, in der das Dokument verÃ¶ffentlicht ist. Mit Hilfe der Ellipsen-SchaltflÃ¤che neben dem Eingabefeld kÃ¶nnen Sie nach einem Speicherort suchen.</p> <p>Falls Sie die URL des WSDL-Dokuments nicht kennen oder wenn (bei Client-Anwendungen) eine Ausfallsicherung unterstÃ¤tzt werden soll, dann klicken Sie auf die SchaltflÃ¤che UDDI durchsuchen, um den UDDI-Browser zu Ã¶ffnen. Beim Import eines WSDL-Dokuments mithilfe des UDDI-Browsers initialisiert dieser den Speicherort im WSDL-Importprogramm und veranlasst das Importprogramm, Quelltext zu erzeugen, der den Speicherort des UDDI-Eintrags fÃ¼r das gesuchte Dokument speichert.</p> <p>Nach der Eingabe eines Dateinamens kÃ¶nnen Sie mit der SchaltflÃ¤che Weiter zur Seite Vorschau wechseln. Sie kÃ¶nnen aber auch auf die SchaltflÃ¤che Optionen klicken und die Verbindungseinstellungen fÃ¼r den Server bereitstellen, auf dem sich das WSDL-Dokument befindet, oder festlegen, wie der Experte Quelltext fÃ¼r die Definitionen in diesem Dokument generiert.</p>

Seite Vorschau

Auf der Seite Vorschau zeigt der Experte eine Vorschau des Quelltextes an, den er fÃ¼r die Definitionen im angegebenen WSDL-Dokument generiert hat. AufgefÃ¼hrt sind nur die Definitionen, die der Experte umsetzen kann. Sie haben nun mehrere MÃ¶glichkeiten: Kehren Sie auf die Seite Quelle zurÃ¼ck, um ein weiteres WSDL-Dokument auswÃ¤hlen. Klicken Sie auf Optionen, um die Verbindungsinformationen zu Ã¤ndern oder zu konfigurieren, wie der Experte Quelltext fÃ¼r die Definitionen im WSDL-Dokument generieren soll. Klicken Sie auf Fertig stellen, wenn der Experte aufrufbare Interfaces und Typdefinitionen definieren und registrieren soll.

Element	Beschreibung
WSDL-Komponenten	WÃ¤hlen Sie das Element aus, dessen generierten Code Sie ansehen mÃ¶chten. Wenn Sie einen Service auswÃ¤hlen, werden alle dafÃ¼r generierten Definitionen angezeigt. Sie kÃ¶nnen aber auch ein einzelnes Service-Element wÃ¤hlen (z.B. ein Interface, eine Methode oder eine Typdefinition).
Code-Vorschau	Zeigt den Code an, den der Experte fÃ¼r das ausgewÃ¤hlte Element generiert hat.
Attribute	Zeigt Informationen Ã¼ber das ausgewÃ¤hlte Element und Einzelheiten Ã¼ber die Definition an, wie z.B. den Namen, den zugehÃ¶rigen Namespace, Namen von Klasse und Typ, die der Experte zur Darstellung eines definierten Typs benutzt hat, die Parameter und SOAP-Aktion einer Methode (Operation), Bindungsinformationen fÃ¼r ein Interface (Port-Typ) usw.

Optionen	Mit dieser SchaltflÃ¤che Ã¶ffnen Sie das Dialogfeld Optionen fÃ¼r den Import, in dem Sie Informationen eingeben kÃ¶nnen, die das Importprogramm benÃ¶tigt, um eine Verbindung mit dem Server herzustellen, der das WSDL-Dokument publiziert, oder um zu konfigurieren, wie der Experte Quelltext generieren soll.
----------	---

2.133 Optionen des Experten für Datenbindungen

Datei ▶ Neu ▶ Weitere ▶ Delphi-Projekte ▶ Neu ▶ XML-Datenbindung ▶ Schaltfläche Optionen

Verwenden Sie dieses Dialogfeld, um festzulegen, wie der Experte für XML-Datenbindung Interfaces und Implementierungsklassen zur Repräsentation eines XML-Dokuments oder -Schemas generieren soll.

Element	Beschreibung	
Kategorie	Wählen Sie eine Optionskategorie, wie z.B. Code-Erzeugung, aus. Die Tabelle rechts neben der Liste zeigt die Optionen der betreffenden Kategorie an.	
Optionentabelle	Bearbeiten Sie die Werte in der zweiten Spalte dieser Tabelle, um die Einstellungen, die der Experte verwendet, zu ändern. Bei der Kategorie Datentypzuweisung zeigt diese Tabelle die Typen an, die der Experte für jeden XML-Typ im Schema generiert. Sie können die angezeigten Werte ändern. Bei dem Datentyp <i>Variant</i> kann die Anwendung beispielsweise zwischen einem leeren String und einem leeren Wert unterscheiden.	
Präfix Get-Methoden (PropGetPrefix)	für	Steuert den Namen, den der Experte Methoden zuweist, die er zum Lesen von Eigenschaftswerten erstellt. Diese Methodennamen bestehen aus dem Präfix Get und dem Namen der Eigenschaft (Element).
Präfix Set-Methoden (PropSetPrefix)	für	Steuert den Namen, den der Experte Methoden zuweist, die er zum Schreiben von Eigenschaftswerten erstellt. Diese Methodennamen bestehen aus dem Präfix Set und dem Namen der Eigenschaft (Element).
Präfix Klassennamens (ClassPrefix)	des	Steuert die Namen, die der Experte den Implementierungsklassen von Knoten zuweist. Der Klassenname besteht aus dem Klassennamenpräfix und dem Namen des Elements oder Attributs.
Präfix des Interface (IntfPrefix)		Steuert die Namen, die der Experte einem Interface zuweist. Der Interface-Name besteht aus dem Interface-Präfix und dem Namen des Elements.
Typsuffix Knotenliste (NodeListSuffix)	der	Steuert die Namen, die der Experte Klassen und Interfaces zuweist, die er für wiederholte Kollektionen untergeordneter Elemente generiert. Die Klassennamen bestehen aus dem Klassennamenpräfix bzw. Interface-Präfix, dem Tag-Namen des untergeordneten Knotens und dem angegebenen Suffix der Knotenliste.
Interface-Basis Knotens (NodeIntfBase)	des	Gibt das Interface an, das als Basis verwendet wird und von dem alle generierten Knoten-Interfaces abgeleitet werden.
Klassenbasis Knotens (NodeClassBase)	des	Gibt die Basisklasse an, von der alle generierten Implementierungsklassen abgeleitet werden. Diese Klasse sollte das unter NodeIntfBase angegebene Interface implementieren.
Schnittstellenbasis der Sammlung	der	Gibt das Interface an, von dem alle generierten Interfaces abgeleitet werden, die wiederholte untergeordnete Knoten repräsentieren.
Klassenbasis Kollektion (CollClassBase)	der	Gibt die Klasse an, von der alle Klassen abgeleitet werden, die wiederholte untergeordnete Knoten repräsentieren.
Standard-Datentyp (DefDataType)		Gibt den Typ an, der den Knoten auf der zweiten Seite des Experten standardmäßig zugewiesen wird.

2.134 Experte für XML-Datenbindung, Seite 1

Datei ▶ **Neu** ▶ **Weitere** ▶ **Delphi-Projekte** ▶ **Neu** ▶ **XML-Datenbindung**

Verwenden Sie diesen Experten zum Erzeugen von Interface- und Klassendefinitionen, die der Struktur eines XML-Dokuments oder -Schemas entsprechen. Der Experte generiert eine globale Funktion, die das Interface für das Root-Element des Dokuments zurückgibt.

Nachdem Sie diese Definitionen mit dem Experten erstellt haben, können Sie mithilfe der Klassen und Interfaces XML-Dokumente bearbeiten, die die Struktur des angegebenen Dokuments oder Schemas haben.

Element	Beschreibung
Schema- XML-Datendatei oder	Geben Sie den Dateinamen des Schemas oder XML-Dokuments ein, für das der Experte Interfaces und Implementierungsklassen erstellen soll. Mithilfe der Ellipsen-Schaltfläche (...) neben dem Eingabefeld können Sie nach einem XML-Dokument oder einer Schemadatei suchen.
XDB-Einstellungsdatei verwenden	Legt fest, ob der Experte mit den zuletzt von Ihnen gespeicherten Einstellungen initialisiert werden soll. Ist das Kontrollkästchen aktiviert, verwendet der Experte die Einstellungen, die Sie in der letzten Sitzung auf der dritten Seite des Experten gespeichert haben.
Optionen	Zeigt das Dialogfeld Optionen des Experten für Datenbindungen an. Sie können verschiedene Optionen auswählen, die festlegen, wie der Experte den Code für die Interfaces und die Implementierungsklassen in Ihrem XML-Dokument oder -Schema generiert.

2.135 Experte für XML-Datenbindung, Seite 2

[Datei](#) ▶ [Neu](#) ▶ [Weitere](#) ▶ [Delphi-Projekte](#) ▶ [Neu](#) ▶ [XML-Datenbindung](#)

Verwenden Sie diese Seite des Experten, um festzulegen, welchen Code der Experte generiert.

Element	Beschreibung
Schema-Komponenten	Zeigt eine Hierarchie der Elemente an, für die der Experte Interfaces und Klassen generieren kann. Diese Hierarchie ist in komplexe Elemente (Knoten mit Tags für untergeordnete Knoten) und einfache Elemente (einfache Datentypen für Elemente im XML-Dokument) unterteilt. Wenn Sie auf einen komplexen Knoten klicken, werden die Knoten für die untergeordneten Elemente eingeblendet. Wenn Sie unter Schema-Komponenten einen Knoten auswählen, werden in der rechten Hälfte des Dialogfeldes detaillierte Informationen über den Knoten angezeigt. Hier können Sie auch festlegen, welchen Quelltext der Experte für diesen Knoten generieren soll.
Quellname	Zeigt den Namen des Typs oder Tags im XML-Schema an. Bearbeiten Sie den Wert, wenn der Experte die Schemadatei erstellen oder ändern soll.
Quelldatentyp	Zeigt den Typ des ausgewählten Knotens an, wie er im XML-Schema definiert ist. Bearbeiten Sie den Wert, wenn der Experte die Schemadatei erstellen oder ändern soll.
Dokumentation	Zeigt Kommentare aus dem XML-Schema an, die den Typ oder Knoten beschreiben. Bearbeiten Sie den Wert, wenn der Experte die Schemadatei erstellen oder ändern soll.
Bindung erzeugen	Erstellt ein Interface und eine Implementierungsklasse für einen ausgewählten komplexen Typ oder eine Eigenschaft im übergeordneten Interface und der übergeordneten Klasse für einfache Elemente, die einem komplexen Typ untergeordnet sind.
Bezeichnername	Legt den Namen des Interface fest, das für einen komplexen Typ der obersten Ebene generiert werden soll. Bei einem komplexen Typ untergeordneten Typen legt Bezeichnername den Name der Eigenschaft fest, die in dem Interface des übergeordneten Elements erzeugt wird.
Typ des Dokumentelements	Diese Option steht nur für komplexe Typen der obersten Ebene zur Verfügung und gibt an, welchen Typ das Dokumentelement (das Root der Datenhierarchie) besitzt.
Elementname	Gibt den Tag-Name des Dokumentelements an.
Datentyp	Bei untergeordneten Knoten können Sie den Typ der Eigenschaft angeben, die das Element repräsentiert. Hat das Element seinerseits untergeordnete Knoten, können Sie in der Dropdown-Liste aus allen Schnittstellentypen wählen, die der Experte für komplexe Typen generieren kann. Bei einfachen Elementen enthält die Dropdown-Liste Typen wie <i>Integer</i> , <i>String</i> , <i>Variant</i> usw. Wenn Sie einfache untergeordnete Elemente als Varianten repräsentieren, hat die Anwendung die Möglichkeit, zwischen Leerstrings und nicht angezeigten Elementen (Null-Varianten) zu unterscheiden.
Wiederholend	Gibt bei untergeordneten Elementen, die komplexe Typen repräsentieren, an, ob der übergeordnete Knoten mehr als einen untergeordneten Knoten dieses Typs haben kann.
Zugriffsmodus	Gibt bei untergeordneten Knoten, die einfache Elemente (keine komplexen Typen) repräsentieren, an, ob die generierte Eigenschaft schreibgeschützt ist oder nicht.
Nativer Typ	Legt, wenn ein einfacher Typ ausgewählt ist, den Datentyp fest, mit dem der Experte Werte dieses Typs repräsentiert.
Optionen	Zeigt das Dialogfeld Optionen des Experten für Datenbindungen an. Sie können verschiedene Optionen auswählen, die festlegen, wie der Experte den Code für die Interfaces und die Implementierungsklassen in Ihrem XML-Dokument oder -Schema generiert.

2.136 Experte für XML-Datenbindung, Seite 3

[Datei](#) ▶ [Neu](#) ▶ [Weitere](#) ▶ [Delphi-Projekte](#) ▶ [Neu](#) ▶ [XML-Datenbindung](#)

Verwenden Sie diese Seite des Experten, um Ihre Auswahl zu bestätigen, Unit-bezogene Optionen für die Code-Generierung festzulegen, Ihre Einstellungen zu speichern und den Experten zu veranlassen, den Code zur Repräsentation des XML-Dokuments oder -Schemas zu generieren.

Element	Beschreibung	
Erzeugte Schnittstellen	Gibt an, welche Interfaces der Experte erzeugen wird. Wenn Sie ein Interface markieren, sehen Sie im Bereich Code-Vorschau die Interface-Definition, die der Experte generieren wird.	
Code-Vorschau	Zeigt den Code an, den der Experte für das unter Generierte Interfaces ausgewählte Interface generiert.	
Einstellungen speichern	nicht	Erzeugt den gewünschten Code, speichert aber Ihre Festlegungen nicht.
Im XML-Schema speichern	Aktualisiert die Schemadatei mit Informationen über die getroffenen Festlegungen.	
In Datei speichern	Geben Sie den Namen der Schemadatei ein, in der der Experte die Informationen über die getroffenen Festlegungen speichern soll. Diese Schemadatei ist unabhängig von dem XML-Dokument oder der Schemadatei, die Sie auf der ersten Seite des Experten ausgewählt haben. Der Experte verwendet diese Datei, um sich selbst beim nächsten Aufruf zu initialisieren.	
Optionen	Zeigt das Dialogfeld Optionen des Experten für Datenbindungen an. Sie können verschiedene Optionen auswählen, die festlegen, wie der Experte den Code für die Interfaces und die Implementierungsklassen in Ihrem XML-Dokument oder -Schema generiert.	
Fertig stellen	Schließt den Experten und erzeugt die Interfaces und Implementierungsklassen für Ihr XML-Dokument oder -Schema.	

2.137 A (Anker) HTML-Element

Das A-Element (Anker) weist den Beginn oder das Ziel eines Hypertext-Links aus. Für das Anker-Element muss das Attribut HREF= oder NAME= festgelegt werden. Es stellt die folgenden Attribute und Ereignisse bereit:

Element	Beschreibung
accesskey	Weist einem Element einen Zugriffsschlüssel zu. Ein Zugriffsschlüssel ist eine einzelnes Zeichen aus dem Zeichensatz des Dokuments. Autoren sollten beim Festlegen eines Zugriffsschlüssels die Eingabemethode des wahrscheinlichen Readers beachten.
charset	Legt die Zeichencodierung der Ziel-Ressource fest.
class	Weist einem Element einen Klassennamen oder eine Klassennamengruppe zu. Es können beliebig viele Elemente demselben Klassennamen zugewiesen werden. Mehrere Klassennamen müssen durch Leerzeichen getrennt werden.
coords	Legt die Position und die Form auf dem Bildschirm relativ zu der oberen, linken Ecke des Objekts fest. Die Anzahl und Reihenfolge der Werte hängt von der Form ab, die definiert werden soll. Mögliche Kombinationen: rect left-x, top-y, right-x, bottom-y. circle center-x, center-y, radius. Wenn der Radius-Wert ein Prozentwert ist, sollte der endgültige Radius von Benutzer-Agents auf der Basis der Breite und Höhe des zugeordneten Objekts berechnet werden. Der Radius sollte der kleinere Wert sein. poly x1, y1, x2, y2, ..., xN, yN. Das erste x- und y-Koordinatenpaar und das letzte sollten gleich sein, um das Polygon zu schließen. Sind diese Koordinatenwerte nicht gleich, sollten Benutzer-Agents ein zusätzliches Koordinatenpaar zum Schließen des Polygons bereitstellen.
dir	Legt die Richtung von direktionalem, neutralem Text in dem Elementinhalt und den Attributwerten fest: ltr Text oder Tabelle von links nach rechts. rtl Text oder Tabelle von rechts nach links.
href	Gibt den Speicherort einer Webressource an und definiert einen Link zwischen dem aktuellen Element (dem Quellanker) und dem Zielanker, den dieses Attribut definiert.
hreflang	Legt die Basissprache der Zielressource fest und kann nur verwendet werden, wenn href definiert ist.
id	Weist einem Element einen Namen zu. Dieser Name muss in einem Dokument eindeutig sein.
lang	Legt die Basissprache der Attributwerte und des Textinhalts eines Elements fest. Der Standardwert für dieses Attribut ist unknown.
name	Weist einen Namen zu.
onblur	Das Ereignis onblur tritt ein, wenn ein Element den Fokus entweder durch ein Zeigegerät oder durch die Taste Tab verliert.
onclick	Das Ereignis onclick tritt ein, wenn eine Taste des Zeigegeräts auf einem Element gedrückt wird.
ondblclick	Das Ereignis ondblclick tritt ein, wenn eine Taste des Zeigegeräts auf einem Element zweimal gedrückt wird.
onfocus	Das Ereignis onfocus tritt ein, wenn ein Element den Fokus entweder durch ein Zeigegerät oder durch die Taste Tab erhält.
onkeydown	Das Ereignis onkeydown tritt ein, wenn eine Taste auf einem Element gedrückt wird.
onkeypress	Das Ereignis onkeypress tritt ein, wenn eine Taste auf einem Element gedrückt und wieder losgelassen wird.

onkeyup	Das Ereignis onkeyup tritt ein, wenn eine Taste auf einem Element losgelassen wird. Dieses Attribut kann für die meisten Elemente verwendet werden.
onmousedown	Das Ereignis onmousedown tritt ein, wenn eine Taste des Zeigegeräts auf einem Element gedrückt wird.
onmousemove	Das Ereignis onmousemove tritt ein, wenn das Zeigegerät über einem Element bewegt wird.
onmouseout	Das Ereignis onmouseout tritt ein, wenn das Zeigegerät aus einem Element bewegt wird.
onmouseover	Das Ereignis onmouseover tritt ein, wenn das Zeigegerät über einem Element bewegt wird.
onmouseup	Das Ereignis onmouseup tritt ein, wenn eine Taste des Zeigegeräts auf einem Element losgelassen wird.
rel	Gibt die Beziehung des aktuellen Dokuments zu dem im Attribut href festgelegten Anker an. Der Wert dieses Attributs ist eine durch Leerzeichen getrennte Liste mit Link-Typen.
rev	Gibt den umgekehrten Link von dem im Attribut href festgelegten Anker zu dem aktuellen Dokument an. Der Wert dieses Attributs ist eine durch Leerzeichen getrennte Liste mit Link-Typen.
shape	Wenn das Attribut type auf image gesetzt ist, gibt dieses Attribut den Speicherort des Bildes an, das für die Schaltfläche Übertragen verwendet werden soll.
style	Legt die Formatinformationen für das aktuelle Element fest. Die Syntax des Wertes des Attributs style hängt von der Standard-Stylesheet-Sprache ab.
tabindex	Legt die Position des aktuellen Elements in der Tabulator-Reihenfolge des aktuellen Dokuments fest. Dieser Wert muss zwischen 0 und 32767 liegen. Benutzer-Agenten sollten führende Nullen ignorieren.
title	Legt Informationen über das Element, für das das Attribut gesetzt wird, fest.
type	Stellt einen Hinweis über den Inhaltstyp des an der Link-Zieladresse verfügbaren Inhalts bereit. Wenn der Hinweis Benutzer-Agents darauf hinweist, dass sie den Inhaltstyp nicht unterstützen, können sie einen Fallback-Mechanismus einsetzen, anstatt den Inhalt abzuholen. Der hier angegebene Inhaltstyp und der tatsächliche Inhalt an der Link-Zieladresse können inkonsistent werden. Autoren, die dieses Attribut einsetzen, müssen dafür Vorsorge treffen. Eine aktuelle Liste der registrierten Inhaltstypen finden Sie auf der Website www.w3.org .

Siehe auch

[W3C HTML-Homepage](http://www.w3.org)

2.138 Unit

Verwenden Sie dieses Dialogfeld zum Festlegen eines Wertes und einer Maßeinheit für das im Fenster Code-Vervollständigung ausgewählte Attribut CSS.

Element	Beschreibung
Wert	Wählen Sie eine Einheitenanzahl für das Attribut aus.
Einheiten	Wählen Sie eine der folgenden Maßeinheiten für das Attribut aus: em entspricht der Schriftgröße des aktuellen Elements. ex annähernd die halbe Höhe der Schriftgröße. px Pixel in Zoll cm Zentimeter mm Millimeter pt Punkt (ein Punkt ist 1/72 eines Zolls) pc Pica (ungefähr 12 Punkt)

2.139 DIV HTML-Element

Das Element DIV wird mit dem Attribut CLASS= verwendet, um verschiedene Arten von Containern zu repräsentieren. Es stellt die folgenden Attribute und Ereignisse bereit:

Element	Beschreibung
align	Legt die horizontale Ausrichtung des Elements in bezug auf den umgebenden Kontext fest. Mögliche Werte sind: left , right , center und justify .
class	Weist einem Element einen Klassennamen oder eine Klassennamengruppe zu. Es können beliebig viele Elemente demselben Klassennamen zugewiesen werden. Mehrere Klassennamen müssen durch Leerzeichen getrennt werden.
dir	Legt die Richtung von direktionalem, neutralem Text in dem Elementinhalt und den Attributwerten fest: ltr Text oder Tabelle von links nach rechts. rtl Text oder Tabelle von rechts nach links.
id	Weist einem Element einen Namen zu. Dieser Name muss in einem Dokument eindeutig sein.
lang	Legt die Basissprache der Attributwerte und des Textinhalts eines Elements fest. Der Standardwert für dieses Attribut ist unknown.
onclick	Das Ereignis onclick tritt ein, wenn eine Taste des Zeigegeräts auf einem Element gedrückt wird.
ondblclick	Das Ereignis ondblclick tritt ein, wenn eine Taste des Zeigegeräts auf einem Element zweimal gedrückt wird.
onkeydown	Das Ereignis onkeydown tritt ein, wenn eine Taste auf einem Element gedrückt wird.
onkeypress	Das Ereignis onkeypress tritt ein, wenn eine Taste auf einem Element gedrückt und wieder losgelassen wird.
onkeyup	Das Ereignis onkeyup tritt ein, wenn eine Taste auf einem Element losgelassen wird. Dieses Attribut kann für die meisten Elemente verwendet werden.
onmousedown	Das Ereignis onmousedown tritt ein, wenn eine Taste des Zeigegeräts auf einem Element gedrückt wird.
onmousemove	Das Ereignis onmousemove tritt ein, wenn das Zeigegerät über einem Element bewegt wird.
onmouseout	Das Ereignis onmouseout tritt ein, wenn das Zeigegerät aus einem Element bewegt wird.
onmouseover	Das Ereignis onmouseover tritt ein, wenn das Zeigegerät über einem Element bewegt wird.
onmouseup	Das Ereignis onmouseup tritt ein, wenn eine Taste des Zeigegeräts auf einem Element losgelassen wird.
style	Legt die Formatinformationen für das aktuelle Element fest. Die Syntax des Wertes des Attributs style hängt von der Standard-Stylesheet-Sprache ab.
title	Legt Informationen über das Element, für das das Attribut gesetzt wird, fest.

Siehe auch

[W3C HTML-Homepage](#)

2.140 HR HTML-Element

Das Element HR zeichnet in einem Dokument eine horizontale Linie. Es stellt die folgenden Attribute und Ereignisse bereit:

Element	Beschreibung
align	Legt die horizontale Ausrichtung des Elements in bezug auf den umgebenden Kontext fest. Mögliche Werte sind: left , right , center und justify .
class	Weist einem Element einen Klassennamen oder eine Klassennamengruppe zu. Es können beliebig viele Elemente demselben Klassennamen zugewiesen werden. Mehrere Klassennamen müssen durch Leerzeichen getrennt werden.
dir	Legt die Richtung von direktionalem, neutralem Text in dem Elementinhalt und den Attributwerten fest: ltr Text oder Tabelle von links nach rechts. rtl Text oder Tabelle von rechts nach links.
id	Weist einem Element einen Namen zu. Dieser Name muss in einem Dokument eindeutig sein.
lang	Legt die Basissprache der Attributwerte und des Textinhalts eines Elements fest. Der Standardwert für dieses Attribut ist unknown .
noshade	Bei true wird die horizontale Linie ausgefüllt dargestellt. Bei false wird die horizontale Linie als zweifarbig Rille dargestellt.
onclick	Das Ereignis onclick tritt ein, wenn eine Taste des Zeigegeräts auf einem Element gedrückt wird.
ondblclick	Das Ereignis ondblclick tritt ein, wenn eine Taste des Zeigegeräts auf einem Element zweimal gedrückt wird.
onkeydown	Das Ereignis onkeydown tritt ein, wenn eine Taste auf einem Element gedrückt wird.
onkeypress	Das Ereignis onkeypress tritt ein, wenn eine Taste auf einem Element gedrückt und wieder losgelassen wird.
onkeyup	Das Ereignis onkeyup tritt ein, wenn eine Taste auf einem Element losgelassen wird. Dieses Attribut kann für die meisten Elemente verwendet werden.
onmousedown	Das Ereignis onmousedown tritt ein, wenn eine Taste des Zeigegeräts auf einem Element gedrückt wird.
onmousemove	Das Ereignis onmousemove tritt ein, wenn das Zeigegerät über einem Element bewegt wird.
onmouseout	Das Ereignis onmouseout tritt ein, wenn das Zeigegerät aus einem Element bewegt wird.
onmouseover	Das Ereignis onmouseover tritt ein, wenn das Zeigegerät über einem Element bewegt wird.
onmouseup	Das Ereignis onmouseup tritt ein, wenn eine Taste des Zeigegeräts auf einem Element losgelassen wird.
size	Legt die Anfangsbreite des Elements fest. Die Breite wird in Pixel angegeben, außer wenn das Attribut type auf text oder password gesetzt ist. In diesem Fall bezieht sich Wert auf die (ganzzahlige) Anzahl der Zeichen.
src	Wenn das Attribut type auf image gesetzt ist, gibt dieses Attribut den Speicherort des Bildes an, das für die Schaltfläche Übertragen verwendet werden soll.
style	Legt die Formatinformationen für das aktuelle Element fest. Die Syntax des Wertes des Attributs style hängt von der Standard-Stylesheet-Sprache ab.
title	Legt Informationen über das Element, für das das Attribut gesetzt wird, fest.

width	Legt die Breite der Linie fest. Die Vorgabe ist 100 %, was bedeutet, dass die Linie über die gesamte Zeichenfläche gezeichnet wird.
-------	---

Siehe auch

[W3C HTML-Homepage](#)

2.141 IMG HTML-Element

Das Element IMGbettet ein Bild oder einen Videoclip in das Dokument ein. Es stellt die folgenden Attribute und Ereignisse bereit:

Element	Beschreibung
align	Legt die horizontale Ausrichtung des Elements in bezug auf den umgebenden Kontext fest. Mögliche Werte sind: left , right , center und justify .
alt	Legt für Benutzer-Agents, die keine Bilder, Formulare oder Applets anzeigen können, alternativen Text fest. Die Sprache des alternativen Texts wird von dem Attribut lang festgelegt.
border	Gibt die Breite des Bildrandes in Pixel an.
class	Weist einem Element einen Klassennamen oder eine Klassennamengruppe zu. Es können beliebig viele Elemente demselben Klassennamen zugewiesen werden. Mehrere Klassennamen müssen durch Leerzeichen getrennt werden.
dir	Legt die Richtung von direktionalem, neutralem Text in dem Elementinhalt und den Attributwerten fest: ltr Text oder Tabelle von links nach rechts. rtl Text oder Tabelle von rechts nach links.
height	Legt die Höhe des Bildes fest.
hspace	Legt den Leerraum fest, der links und rechts neben dem Bild eingefügt werden soll.
id	Weist einem Element einen Namen zu. Dieser Name muss in einem Dokument eindeutig sein.
ismap	Ordnet bei IMG- und INPUT-Elementen dem Element eine serverseitige Imagemap zu.
lang	Legt die Basissprache der Attributwerte und des Textinhalts eines Elements fest. Der Standardwert für dieses Attribut ist unknown.
longdesc	Legt einen Link zu einer langen Beschreibung des Bildes fest.
name	Weist einen Namen zu.
onclick	Das Ereignis onclick tritt ein, wenn eine Taste des Zeigegeräts auf einem Element gedrückt wird.
ondblclick	Das Ereignis ondblclick tritt ein, wenn eine Taste des Zeigegeräts auf einem Element zweimal gedrückt wird.
onkeydown	Das Ereignis onkeydown tritt ein, wenn eine Taste auf einem Element gedrückt wird.
onkeypress	Das Ereignis onkeypress tritt ein, wenn eine Taste auf einem Element gedrückt und wieder losgelassen wird.
onkeyup	Das Ereignis onkeyup tritt ein, wenn eine Taste auf einem Element losgelassen wird. Dieses Attribut kann für die meisten Elemente verwendet werden.
onmousedown	Das Ereignis onmousedown tritt ein, wenn eine Taste des Zeigegeräts auf einem Element gedrückt wird.
onmousemove	Das Ereignis onmousemove tritt ein, wenn das Zeigegerät über einem Element bewegt wird.
onmouseout	Das Ereignis onmouseout tritt ein, wenn das Zeigegerät aus einem Element bewegt wird.
onmouseover	Das Ereignis onmouseover tritt ein, wenn das Zeigegerät über einem Element bewegt wird.
onmouseup	Das Ereignis onmouseup tritt ein, wenn eine Taste des Zeigegeräts auf einem Element losgelassen wird.
src	Gibt den Speicherort des für das Element zu verwendenden Bildes an.

style	Legt die Formatinformationen für das aktuelle Element fest. Die Syntax des Wertes des Attributs style hängt von der Standard-Stylesheet-Sprache ab.
title	Legt Informationen über das Element, für das das Attribut gesetzt wird, fest.
usemap	Ordnet einem Element eine Imagemap zu. Die Imagemap wird von dem Element MAP definiert. Der Wert von usemap muss mit dem Wert des Attributs name des zugeordneten MAP-Elements übereinstimmen.
vspace	Legt den Leerraum fest, der ober- und unterhalb des Bilds eingefügt werden soll.
width	Legt die Breite des Bildes fest.

Siehe auch

[W3C HTML-Homepage](#)

2.142 INPUT HTML-Element

Das Element INPUT legt ein Formulareingabefeld fest. Es stellt die folgenden Attribute und Ereignisse bereit:

Element	Beschreibung
accept	Legt eine durch Komma getrennte Liste mit Inhaltstypen fest, die ein Server, der dieses Formular verarbeitet, korrekt behandeln kann. Benutzer-Agents können anhand dieser Information nicht-konforme Dateien herausfiltern, wenn ein Benutzer Dateien für die Übermittlung zum Server auswählt.
accesskey	Weist einem Element einen Zugriffsschlüssel zu. Ein Zugriffsschlüssel ist eine einzelnes Zeichen aus dem Zeichensatz des Dokuments. Autoren sollten beim Festlegen eines Zugriffsschlüssels die Eingabemethode des wahrscheinlichen Readers beachten.
align	Legt die horizontale Ausrichtung des Elements in bezug auf den umgebenden Kontext fest. Mögliche Werte sind: left , right , center und justify .
alt	Legt für Benutzer-Agents, die keine Bilder, Formulare oder Applets anzeigen können, alternativen Text fest. Die Sprache des alternativen Texts wird von dem Attribut lang festgelegt.
checked	Wenn das Attribut type den Wert radio oder checkbox hat, legt dieses Boolesche Attribut fest, dass das Feld markiert ist. Benutzer-Agents müssen dieses Attribut für andere Steuerelementtypen ignorieren.
class	Weist einem Element einen Klassennamen oder eine Klassennamengruppe zu. Es können beliebig viele Elemente demselben Klassennamen zugewiesen werden. Mehrere Klassennamen müssen durch Leerzeichen getrennt werden.
dir	Legt die Richtung von direktionalem, neutralem Text in dem Elementinhalt und den Attributwerten fest: ltr Text oder Tabelle von links nach rechts. rtl Text oder Tabelle von rechts nach links.
disabled	Wenn das Attribut für ein Formularsteuerelement gesetzt wird, akzeptiert das Steuerelement keine Benutzereingaben.
id	Weist einem Element einen Namen zu. Dieser Name muss in einem Dokument eindeutig sein.
ismap	Ordnet bei IMG- und INPUT-Elementen dem Element eine serverseitige Imagemap zu.
lang	Legt die BasisSprache der Attributwerte und des Textinhalts eines Elements fest. Der Standardwert für dieses Attribut ist unknown .
maxlength	Wenn das Attribut type den Wert text oder password hat, legt dieses Attribut die maximale Zeichenanzahl fest, die ein Benutzer eingeben kann. Die Anzahl kann die festgelegte Größe übersteigen. In diesem Fall sollte der Benutzer-Agent einen Blätter-Mechanismus bereitstellen. Der Vorgabewert für diese Attribut ist eine unbegrenzte Anzahl.
name	Weist einen Namen zu.
onblur	Das Ereignis onblur tritt ein, wenn ein Element den Fokus entweder durch ein Zeigegerät oder durch die Taste Tab verliert.
onchange	Das Ereignis onchange tritt ein, wenn ein Steuerelement den Eingabefokus verliert und sein Wert, während es den Fokus hatte, geändert wurde. Dieses Attribut kann für die folgenden Elemente angewendet werden: INPUT, SELECT und TEXTAREA.
onclick	Das Ereignis onclick tritt ein, wenn eine Taste des Zeigegeräts auf einem Element gedrückt wird.
ondblclick	Das Ereignis ondblclick tritt ein, wenn eine Taste des Zeigegeräts auf einem Element zweimal gedrückt wird.

onfocus	Das Ereignis onfocus tritt ein, wenn ein Element den Fokus entweder durch ein Zeigegerät oder durch die Taste Tab erhält.
onkeydown	Das Ereignis onkeydown tritt ein, wenn eine Taste auf einem Element gedrückt wird.
onkeypress	Das Ereignis onkeypress tritt ein, wenn eine Taste auf einem Element gedrückt und wieder losgelassen wird.
onkeyup	Das Ereignis onkeyup tritt ein, wenn eine Taste auf einem Element losgelassen wird. Dieses Attribut kann für die meisten Elemente verwendet werden.
onmousedown	Das Ereignis onmousedown tritt ein, wenn eine Taste des Zeigegeräts auf einem Element gedrückt wird.
onmousemove	Das Ereignis onmousemove tritt ein, wenn das Zeigegerät über einem Element bewegt wird.
onmouseout	Das Ereignis onmouseout tritt ein, wenn das Zeigegerät aus einem Element bewegt wird.
onmouseover	Das Ereignis onmouseover tritt ein, wenn das Zeigegerät über einem Element bewegt wird.
onmouseup	Das Ereignis onmouseup tritt ein, wenn eine Taste des Zeigegeräts auf einem Element losgelassen wird.
onselect	Das Ereignis onselect tritt ein, wenn ein Benutzer Text in einem Textfeld markiert. Dieses Attribut kann mit den Elementen INPUT und TEXTAREA verwendet werden.
readonly	Wenn dieses Attribut für ein Formularsteuerelement gesetzt ist, kann das Steuerelement nicht geändert werden.
size	Legt die Anfangsbreite des Elements fest. Die Breite wird in Pixel angegeben, außer wenn das Attribut type auf text oder password gesetzt ist. In diesem Fall bezieht sich Wert auf die (ganzzahlige) Anzahl der Zeichen.
src	Wenn das Attribut type auf image gesetzt ist, gibt dieses Attribut den Speicherort des Bildes an, das für die Schaltfläche Übertragen verwendet werden soll.
style	Legt die Formatinformationen für das aktuelle Element fest. Die Syntax des Wertes des Attributs style hängt von der Standard-Stylesheet-Sprache ab.
tabindex	Legt die Position des aktuellen Elements in der Tabulator-Reihenfolge des aktuellen Dokuments fest. Dieser Wert muss zwischen 0 und 32767 liegen. Benutzer-Agenten sollten führende Nullen ignorieren.
title	Legt Informationen über das Element, für das das Attribut gesetzt wird, fest.
type	Legt den Typ des Steuerelements fest und kann auf folgende Werte gesetzt werden: button erstellt eine Schaltfläche. Benutzer-Agenten sollten den Wert des Attributs value als Beschriftung der Schaltfläche verwenden. checkbox erstellt ein Kontrollfeld. file erstellt ein Steuerelement zur Dateiauswahl. Benutzer-Agenten können den Wert des Attributs value als Anfangs-Dateinamen verwenden. hidden erstellt ein verborgenes Steuerelement. image erstellt eine grafische Übertragen-Schaltfläche. Der Wert des Attributs scr gibt die URI des Bildes an, das auf der Schaltfläche angezeigt werden soll. Autoren sollten alternativen Text im Attribut alt für das Bild vorsehen. password erstellt ein einzeiliges Eingabefeld, der Eingabetext wird aber verschlüsselt angezeigt (z.B. als eine Reihe von Sternchen), damit die Zeichen nicht lesbar sind. Dieser Steuerelementtyp wird häufig für sensible Eingaben, wie z.B. Passwörter, verwendet. Beachten Sie, dass der aktuelle Wert der Text ist, der vom Benutzer eingegeben wurde und nicht der vom Benutzer-Agent gerenderte Text. radio erstellt ein Optionsfeld. reset erstellt eine Zurücksetzen-Schaltfläche. submit erstellt eine Übertragen-Schaltfläche. text erstellt ein einzeiliges Eingabefeld.

usemap	Ordnet einem Element eine Imagemap zu. Die Imagemap wird von dem Element MAP definiert. Der Wert von usemap muss mit dem Wert des Attributs name des zugeordneten MAP-Elements übereinstimmen.
value	Legt den Anfangswert des Steuerelements fest. Wenn das Attribut type den Wert radio oder checkbox hat, ist dieses Attribut optional.

Siehe auch

[W3C HTML-Homepage](#)

2.143 SELECT HTML-Element

Das Element SELECT legt ein Listenfeld oder eine Dropdown-Liste fest. Es stellt die folgenden Attribute und Ereignisse bereit:

Element	Beschreibung
class	Weist einem Element einen Klassennamen oder eine Klassennamengruppe zu. Es können beliebig viele Elemente demselben Klassennamen zugewiesen werden. Mehrere Klassennamen müssen durch Leerzeichen getrennt werden.
dir	Legt die Richtung von direktionalem, neutralem Text in dem Elementinhalt und den Attributwerten fest: ltr Text oder Tabelle von links nach rechts. rtl Text oder Tabelle von rechts nach links.
disabled	Wenn das Attribut für ein Formularsteuerelement gesetzt wird, akzeptiert das Steuerelement keine Benutzereingaben.
id	Weist einem Element einen Namen zu. Dieser Name muss in einem Dokument eindeutig sein.
lang	Legt die Basissprache der Attributwerte und des Textinhalts eines Elements fest. Der Standardwert für dieses Attribut ist unknown.
mehrere	Bei true können in der Dropdown-Liste mehrere Einträge ausgewählt werden. Bei false lässt das SELECT-Element nur die Auswahl eines Eintrags zu.
name	Weist einen Namen zu.
onblur	Das Ereignis onblur tritt ein, wenn ein Element den Fokus entweder durch ein Zeigegerät oder durch die Taste Tab verliert.
onchange	Das Ereignis onchange tritt ein, wenn ein Steuerelement den Eingabefokus verliert und sein Wert, während es den Fokus hatte, geändert wurde. Dieses Attribut kann für die folgenden Elemente angewendet werden: INPUT, SELECT und TEXTAREA.
onclick	Das Ereignis onclick tritt ein, wenn eine Taste des Zeigegeräts auf einem Element gedrückt wird.
ondblclick	Das Ereignis ondblclick tritt ein, wenn eine Taste des Zeigegeräts auf einem Element zweimal gedrückt wird.
onfocus	Das Ereignis onfocus tritt ein, wenn ein Element den Fokus entweder durch ein Zeigegerät oder durch die Taste Tab erhält.
onkeydown	Das Ereignis onkeydown tritt ein, wenn eine Taste auf einem Element gedrückt wird.
onkeypress	Das Ereignis onkeypress tritt ein, wenn eine Taste auf einem Element gedrückt und wieder losgelassen wird.
onkeyup	Das Ereignis onkeyup tritt ein, wenn eine Taste auf einem Element losgelassen wird. Dieses Attribut kann für die meisten Elemente verwendet werden.
onmousedown	Das Ereignis onmousedown tritt ein, wenn eine Taste des Zeigegeräts auf einem Element gedrückt wird.
onmousemove	Das Ereignis onmousemove tritt ein, wenn das Zeigegerät über einem Element bewegt wird.
onmouseout	Das Ereignis onmouseout tritt ein, wenn das Zeigegerät aus einem Element bewegt wird.
onmouseover	Das Ereignis onmouseover tritt ein, wenn das Zeigegerät über einem Element bewegt wird.
onmouseup	Das Ereignis onmouseup tritt ein, wenn eine Taste des Zeigegeräts auf einem Element losgelassen wird.

size	Legt die Anfangsbreite des Elements fest. Die Breite wird in Pixel angegeben, außer wenn das Attribut type auf text oder password gesetzt ist. In diesem Fall bezieht sich Wert auf die (ganzzahlige) Anzahl der Zeichen.
style	Legt die Formatinformationen für das aktuelle Element fest. Die Syntax des Wertes des Attributs style hängt von der Standard-Stylesheet-Sprache ab.
tabindex	Legt die Position des aktuellen Elements in der Tabulator-Reihenfolge des aktuellen Dokuments fest. Dieser Wert muss zwischen 0 und 32767 liegen. Benutzer-Agenten sollten führende Nullen ignorieren.
title	Legt Informationen über das Element, für das das Attribut gesetzt wird, fest.

Siehe auch[W3C HTML-Homepage](#)

2.144 SPAN HTML-Element

Mit dem Element SPAN können Sie eine eigene Rendering-Methode definieren. Es stellt die folgenden Attribute und Ereignisse bereit:

Element	Beschreibung
class	Weist einem Element einen Klassennamen oder eine Klassennamengruppe zu. Es können beliebig viele Elemente demselben Klassennamen zugewiesen werden. Mehrere Klassennamen müssen durch Leerzeichen getrennt werden.
dir	Legt die Richtung von direktionalem, neutralem Text in dem Elementinhalt und den Attributwerten fest: ltr Text oder Tabelle von links nach rechts. rtl Text oder Tabelle von rechts nach links.
id	Weist einem Element einen Namen zu. Dieser Name muss in einem Dokument eindeutig sein.
lang	Legt die Basissprache der Attributwerte und des Textinhalts eines Elements fest. Der Standardwert für dieses Attribut ist unknown.
onclick	Das Ereignis onclick tritt ein, wenn eine Taste des Zeigegeräts auf einem Element gedrückt wird.
ondblclick	Das Ereignis ondblclick tritt ein, wenn eine Taste des Zeigegeräts auf einem Element zweimal gedrückt wird.
onkeydown	Das Ereignis onkeydown tritt ein, wenn eine Taste auf einem Element gedrückt wird.
onkeypress	Das Ereignis onkeypress tritt ein, wenn eine Taste auf einem Element gedrückt und wieder losgelassen wird.
onkeyup	Das Ereignis onkeyup tritt ein, wenn eine Taste auf einem Element losgelassen wird. Dieses Attribut kann für die meisten Elemente verwendet werden.
onmousedown	Das Ereignis onmousedown tritt ein, wenn eine Taste des Zeigegeräts auf einem Element gedrückt wird.
onmousemove	Das Ereignis onmousemove tritt ein, wenn das Zeigegerät über einem Element bewegt wird.
onmouseout	Das Ereignis onmouseout tritt ein, wenn das Zeigegerät aus einem Element bewegt wird.
onmouseover	Das Ereignis onmouseover tritt ein, wenn das Zeigegerät über einem Element bewegt wird.
onmouseup	Das Ereignis onmouseup tritt ein, wenn eine Taste des Zeigegeräts auf einem Element losgelassen wird.
style	Legt die Formatinformationen für das aktuelle Element fest. Die Syntax des Wertes des Attributs style hängt von der Standard-Stylesheet-Sprache ab.
title	Legt Informationen über das Element, für das das Attribut gesetzt wird, fest.

Siehe auch

[W3C HTML-Homepage](#)

2.145 TABLE HTML-Element

Das Element TABLE legt fest, dass der enthaltene Inhalt in Form einer Tabelle mit Zeilen und Spalten organisiert wird. Mit den Elementen TR, TD und TH im Container können Sie Zeilen, Spalten und Zellen erstellen. Es stellt die folgenden Attribute und Ereignisse bereit:

Tip: Wählen Sie im Hauptmenü der IDE **Ansicht>Symbolleisten>HTML-Tabelle**, um die Symbolleiste zum Formatieren von Tabellen einzublenden.

Element	Beschreibung
align	Legt die horizontale Ausrichtung des Elements in bezug auf den umgebenden Kontext fest. Mögliche Werte sind: left , right , center und justify .
bgcolor	Legt die Hintergrundfarbe für die Tabellenzellen fest.
border	Legt die Stärke (in Pixel) des Rahmens um die Tabelle fest.
cellpadding	Legt den Abstand zwischen dem Zellrand und dem Zellinhalt fest. Wenn der Wert dieses Attributs in Pixel angegeben ist, erhalten alle vier Ränder diesen Abstand vom Inhalt. Wenn es sich um einen Prozentwert handelt, erhält der obere und der untere Rand einen speziellen Abstand von dem Inhalt. Dieser Abstand ist ein Prozentwert des verfügbaren vertikalen Platzes. Der Abstand des linken und rechten Randes wird gleich groß auf der Basis des verfügbaren horizontalen Platzes errechnet.
cellspacing	Legt fest, wie viel Platz der Benutzer-Agent zwischen der linken Seite der Tabelle und dem linken Rand der ersten Spalte, dem oberen Rand der Tabelle und dem oberen Rand der ersten Zeile (und entsprechend für den rechten und den unteren Rand der Tabelle) lassen soll.
class	Weist einem Element einen Klassennamen oder eine Klassennamengruppe zu. Es können beliebig viele Elemente demselben Klassennamen zugewiesen werden. Mehrere Klassennamen müssen durch Leerzeichen getrennt werden.
datapagesize	Legt die Anzahl der Datensätze fest, die in einer mit einer Datenquelle verbundenen Tabelle angezeigt werden soll.
dir	Legt die Richtung von direktionalem, neutralem Text in dem Elementinhalt und den Attributwerten fest: ltr Text oder Tabelle von links nach rechts. rtl Text oder Tabelle von rechts nach links.
Frame	
id	Weist einem Element einen Namen zu. Dieser Name muss in einem Dokument eindeutig sein.
lang	Legt die BasisSprache der Attributwerte und des Textinhalts eines Elements fest. Der Standardwert für dieses Attribut ist unknown .
onclick	Das Ereignis onclick tritt ein, wenn eine Taste des Zeigegeräts auf einem Element gedrückt wird.
ondblclick	Das Ereignis ondblclick tritt ein, wenn eine Taste des Zeigegeräts auf einem Element zweimal gedrückt wird.
onkeydown	Das Ereignis onkeydown tritt ein, wenn eine Taste auf einem Element gedrückt wird.
onkeypress	Das Ereignis onkeypress tritt ein, wenn eine Taste auf einem Element gedrückt und wieder losgelassen wird.
onkeyup	Das Ereignis onkeyup tritt ein, wenn eine Taste auf einem Element losgelassen wird. Dieses Attribut kann für die meisten Elemente verwendet werden.

onmousedown	Das Ereignis onmousedown tritt ein, wenn eine Taste des Zeigegeräts auf einem Element gedrückt wird.
onmousemove	Das Ereignis onmousemove tritt ein, wenn das Zeigegerät über einem Element bewegt wird.
onmouseout	Das Ereignis onmouseout tritt ein, wenn das Zeigegerät aus einem Element bewegt wird.
onmouseover	Das Ereignis onmouseover tritt ein, wenn das Zeigegerät über einem Element bewegt wird.
onmouseup	Das Ereignis onmouseup tritt ein, wenn eine Taste des Zeigegeräts auf einem Element losgelassen wird.
rules	Legt fest, welche Linien zwischen Zellen in der Tabelle erscheinen sollen. Die Anzeige der Linien ist vom Benutzer-Agent abhängig. Mögliche Werte: none Keine Linien. Das ist der Vorgabewert. groups Linien werden nur zwischen Zeilengruppen und Spaltengruppen angezeigt. rows Linien werden nur zwischen Zeilen angezeigt. cols Linien werden nur zwischen Spalten angezeigt. all Linien werden zwischen allen Zeilen und Spalten angezeigt.
style	Legt die Formatinformationen für das aktuelle Element fest. Die Syntax des Wertes des Attributs style hängt von der Standard-Stylesheet-Sprache ab.
title	Legt Informationen über das Element, für das das Attribut gesetzt wird, fest.
width	Legt die gewünschte Breite der gesamten Tabelle fest und ist für visuelle Benutzer-Agents vorgesehen. Wenn es sich um einen Prozentwert handelt, ist dieser Wert relativ zu dem horizontalen Platz, der dem Benutzer-Agent zur Verfügung steht. Wenn keine Tabellenbreite angegeben wird, legt der Benutzer-Agent die Breite fest.

Siehe auch

[W3C HTML-Homepage](#)

2.146 TEXTAREA HTML-Element

Das Element TEXTAREA legt ein mehrzeiliges Eingabefeld fest. Es stellt die folgenden Attribute und Ereignisse bereit:

Element	Beschreibung
accesskey	Weist einem Element einen Zugriffsschlüssel zu. Ein Zugriffsschlüssel ist eine einzelnes Zeichen aus dem Zeichensatz des Dokuments. Autoren sollten beim Festlegen eines Zugriffsschlüssels die Eingabemethode des wahrscheinlichen Readers beachten.
class	Weist einem Element einen Klassennamen oder eine Klassennamengruppe zu. Es können beliebig viele Elemente demselben Klassennamen zugewiesen werden. Mehrere Klassennamen müssen durch Leerzeichen getrennt werden.
cols	Legt fest, wie viele Zeichen das Eingabefeld breit sein soll. Benutzern sollte es möglich sein, längere Zeilen einzugeben, so dass Benutzer-Agents einen Blätter-Mechanismus bereitstellen sollten. Benutzer-Agents können auch Textzeilen umbrechen, damit lange Zeilen sichtbar bleiben, ohne dass geblättert werden muss.
dir	Legt die Richtung von direktionalem, neutralem Text in dem Elementinhalt und den Attributwerten fest: ltr Text oder Tabelle von links nach rechts. rtl Text oder Tabelle von rechts nach links.
disabled	Wenn das Attribut für ein Formularsteuerelement gesetzt wird, akzeptiert das Steuerelement keine Benutzereingaben.
id	Weist einem Element einen Namen zu. Dieser Name muss in einem Dokument eindeutig sein.
lang	Legt die Basissprache der Attributwerte und des Textinhalts eines Elements fest. Der Standardwert für dieses Attribut ist unknown.
name	Weist einen Namen zu.
onblur	Das Ereignis onblur tritt ein, wenn ein Element den Fokus entweder durch ein Zeigegerät oder durch die Taste Tab verliert.
onchange	Das Ereignis onchange tritt ein, wenn ein Steuerelement den Eingabefokus verliert und sein Wert, während es den Fokus hatte, geändert wurde. Dieses Attribut kann für die folgenden Elemente angewendet werden: INPUT, SELECT und TEXTAREA.
onclick	Das Ereignis onclick tritt ein, wenn eine Taste des Zeigegeräts auf einem Element gedrückt wird.
ondblclick	Das Ereignis ondblclick tritt ein, wenn eine Taste des Zeigegeräts auf einem Element zweimal gedrückt wird.
onfocus	Das Ereignis onfocus tritt ein, wenn ein Element den Fokus entweder durch ein Zeigegerät oder durch die Taste Tab erhält.
onkeydown	Das Ereignis onkeydown tritt ein, wenn eine Taste auf einem Element gedrückt wird.
onkeypress	Das Ereignis onkeypress tritt ein, wenn eine Taste auf einem Element gedrückt und wieder losgelassen wird.
onkeyup	Das Ereignis onkeyup tritt ein, wenn eine Taste auf einem Element losgelassen wird. Dieses Attribut kann für die meisten Elemente verwendet werden.
onmousedown	Das Ereignis onmousedown tritt ein, wenn eine Taste des Zeigegeräts auf einem Element gedrückt wird.
onmousemove	Das Ereignis onmousemove tritt ein, wenn das Zeigegerät über einem Element bewegt wird.
onmouseout	Das Ereignis onmouseout tritt ein, wenn das Zeigegerät aus einem Element bewegt wird.

onmouseover	Das Ereignis onmousemove tritt ein, wenn das Zeigegerät über einem Element bewegt wird.
onmouseup	Das Ereignis onmouseup tritt ein, wenn eine Taste des Zeigegeräts auf einem Element losgelassen wird.
onselect	Das Ereignis onselect tritt ein, wenn ein Benutzer Text in einem Textfeld markiert. Dieses Attribut kann mit den Elementen INPUT und TEXTAREA verwendet werden.
readonly	Wenn dieses Attribut für ein Formularsteuerelement gesetzt ist, kann das Steuerelement nicht geändert werden.
rows	Gibt die Anzahl der sichtbaren Textzeilen in dem Steuerelement an. Benutzern sollte es möglich sein, mehr Zeilen einzugeben, so dass Benutzer-Agents einen Blätter-Mechanismus bereitstellen sollten.
style	Legt die Formatinformationen für das aktuelle Element fest. Die Syntax des Wertes des Attributs style hängt von der Standard-Stylesheet-Sprache ab.
tabindex	Legt die Position des aktuellen Elements in der Tabulator-Reihenfolge des aktuellen Dokuments fest. Dieser Wert muss zwischen 0 und 32767 liegen. Benutzer-Agenten sollten führende Nullen ignorieren.
title	Legt Informationen über das Element, für das das Attribut gesetzt wird, fest.

Siehe auch[W3C HTML-Homepage](#)

2.147 User Control einfügen

Einfügen ▾ **User Control einfügen**

Verwenden Sie dieses Dialogfeld zum Einfügen eines User Controls in ein Web Form oder eine User Control-Template.

Element	Beschreibung
Durchsuchen	Sucht nach einer .ascx-Datei in einem Verzeichnis.
OK	Fügt die .ascx-Datei Ihrem Projekt hinzu.
Abbrechen	Schließt das Dialogfeld, ohne zu speichern.

2.148 Bild einfügen

Einfügen ▶ Bild

Verwenden Sie dieses Dialogfeld zum Einfügen einer Bilddatei in Ihr ASP.NET Web Form oder in Ihre HTML-Seite. Nach dem Einfügen können Sie die folgenden Bildattribute ändern.

Element	Beschreibung
Ausrichtung	Legt die Position des Bildes oder Objekts in Bezug auf dessen Kontext fest.
Randgröße	Gibt die Breite des Rands für das Bild oder Objekt in Pixel an.
Breite	Gibt die neue Breite für das markierte Bild oder Objekt an.
Horizontaler Abstand	Legt den Leerraum fest, der links und rechts neben dem Bild oder Objekt eingefügt werden soll.
Höhe	Gibt die neue Höhe für das markierte Bild oder Objekt an.
Vertikaler Abstand	Legt den Leerraum fest, der ober- und unterhalb des Bilds oder Objekts eingefügt werden soll.
Alternativer Text	Geben Sie einen alternativen Text ein, der angezeigt wird, wenn das Element nicht normal dargestellt werden kann.
Imagemap	Legt ein client-seitiges Imagemap (oder einen anderen Navigationsmechanismus) fest, das einem anderen Element zugeordnet werden kann.
Imagemap-Informationen auf dem Server	Definiert ein server-seitiges Imagemap für das angegebene Bild.

2.149 Eingabe einfügen

Einfügen ▶ Eingabe

Verwenden Sie dieses Dialogfeld, um vor dem Deployment Steuerelemente in Ihrem ASP.NET Web Form oder Ihrer HTML-Seite zu erstellen oder zu ändern.

Element	Beschreibung
Eingabetyp	Legt den Typ der zu erstellenden Steuerelemente fest.
Name	Gibt den Namen des Formulars an.
Ausrichtung	Legt die horizontale Ausrichtung des Elements in bezug auf den umgebenden Kontext fest.
Wert	Legt den Anfangswert des Steuerelements fest.
Alternativer Text	Legt einen alternativen Text fest, der angezeigt wird, wenn das Element, wie z.B. Bilder oder Applets, nicht normal dargestellt werden kann.
Tab-Index	Legt die Position des aktuellen Elements in der Tabulator-Reihenfolge des aktuellen Dokuments fest. Alle Werte zwischen 0 und 32767 sind zulässig.
Schreibgeschützt	Verhindert, dass das Steuerelement verändert wird.
Maximale Länge	Legt die maximale Zeichenanzahl fest, die eingegeben werden kann, wenn das Attribut einen Text- oder Passwortwert besitzt.

2.150 Tabelle einfügen

Einfügen ▾ Tabelle

Verwenden Sie dieses Dialogfeld zum Einfügen einer Tabelle in Ihr ASP.NET Web Form oder in Ihre HTML-Seite. Nach dem Einfügen können Sie das Erscheinungsbild der Tabelle ändern.

Element	Beschreibung
Zeilen	Gibt die Anzahl der sichtbaren Textzeilen an.
Spalten	Gibt die Anzahl der Spalten an, die in der Tabelle enthalten sein soll.
Breite	Legt die gewünschte Breite für die gesamte Tabelle fest.
Pixel	Legt einen Integerwert fest, der die Anzahl der Pixel auf dem Bildschirm repräsentiert.
Prozent	Legt einen Wert in Prozent (%) fest.
Höhe	Legt die gewünschte Höhe für die gesamte Tabelle in Pixel fest.
Ausrichtung	Legt die Position der Tabelle in bezug auf das Dokument fest.
Hintergrundfarbe	Legt die Hintergrundfarbe für die Tabelle fest.
Hintergrundbild	Legt das Hintergrundbild für die Tabelle fest.
Rahmenfarbe	Legt die Farbe für den Rahmen der Tabelle fest.
Randgröße	Legt die Breite (nur in Pixel) für den Tabellenrahmen fest.
Hervorhebungsfarbe	Legt die Farbe für den hellen Teil des Tabellenrahmens fest.
Zellzwischenraum	Legt fest, wie viel Platz zwischen der linken Seite der Tabelle und dem linken Rand der ersten Spalte, dem oberen Rand der Tabelle und dem oberen Rand der ersten Zeile (und entsprechend für den rechten und den unteren Rand der Tabelle) frei bleiben soll. Das Attribut legt auch fest, wie viel Platz zwischen den einzelnen Zellen frei bleiben soll.
Schattenfarbe	Legt die Farbe für den schattierten Teil des Tabellenrahmens fest.
Innenabstand	Legt den Abstand zwischen dem Zellrand und dem Zellinhalt fest.
Übersicht	Stellt eine Zusammenfassung des Zwecks und der Struktur der Tabelle für die Übergabe an nichtvisuelle Medien bereit.
Hintergrundfarbe Zellattribute	Legt die Hintergrundfarbe für die einzelnen Zellen in einer Tabelle fest.
Rahmenfarbe Zellattribute	Legt die Farbe für den Rahmen der einzelnen Zellen in einer Tabelle fest.
Textausrichtung	Legt die Ausrichtung von Daten und Text in einer Zelle fest.
Text umbrechen	Markieren Sie dieses Feld, damit Text entsprechend der Tabellenbreite umbrochen wird.

2.151 Farbe auswählen

Verwenden Sie dieses Dialogfeld zum Ändern der Vorder- und Hintergrundfarbe im HTML-Designer.

Element	Beschreibung
Web-Palette	Verwenden Sie diese Registerseite zum Auswählen von verfügbaren Farben.
Farbnamen verwenden	Wählen Sie, ob Sie entweder einen Webfarb-String, z.B. #FF00FF, oder einen Webfarbnamen wie z.B. "snow" verwenden möchten. Wenn das Kontrollfeld markiert ist, werden im Dialogfeld - soweit verfügbar - Webfarbnamen verwendet.
Ausgewählte Farbe anpassen	Ermöglicht das Festlegen und Ändern der Werte des Farbschemas, um die Zusammensetzung der ausgewählten Farbe zu ändern.
Benannte Webfarben	Verwenden Sie diese Registerseite zum schnellen Auswählen von benannten Webfarben.
Weitere Farben	Verwenden Sie diese Registerseite zum Auswählen von System- oder Standardfarben.
Systemfarben	Führt verfügbare Farben aus der Windows-Systempalette auf.
Standardfarben	Führt verfügbare Farben aus der Windows-Standard-16-Farbpalette auf.

2.152 EJBs aus Liste auswählen

J2EE-Referenz hinzufügen

Das Dialogfeld EJBs aus Liste auswählen ermöglicht es, eine .NET-Assemblierung aus [.jar](#) oder [.ear](#)-Dateien zu generieren. Sie können die Assemblierung entweder für alle EJBs in einem Archiv erzeugen oder einzelne EJBs aus einer Liste auswählen.

Element	Beschreibung
Assemblierung mit allen EJBs im J2EE-Archiv erzeugen	Die IDE erstellt für alle EJBs im Archiv eine .NET-Assemblierung.
Wählen Sie die EJBs aus, für die eine Assemblierung erzeugt werden soll.	Aktiviert die Liste der im Archiv enthaltenen EJBs.
EJB-Liste	Aktivieren oder deaktivieren Sie dieses Kontrollfeld der einzelnen EJBs, um diese in die Assemblierung einzubeziehen oder auszuschließen.
Alle	Wählt alle EJBs in der Liste aus.
Ohne	Entfernt alle Häkchen aus allen Kontrollfeldern der EJBs in der Liste.

2.153 Testfall-Experte

Datei ▶ **Neu** ▶ **Weitere** ▶ **Unit-Test** ▶ **Testfall**

Verwenden Sie diesen Experten, um einen Testfall für Ihr Projekt zu erstellen.

Element	Beschreibung
Quelldatei	Legt den Pfad und den Dateinamen für die Quelltextdatei fest, für die Tests hinzugefügt werden sollen. Dieses Textfeld enthält per Vorgabe den Namen der Hauptquelltextdatei Ihres aktiven Projekts. Sie können das Verzeichnis und den Namen der Datei bei Bedarf ändern. Wenn die Datei nicht vorhanden ist, können Sie keine Klassen und Methoden zum Testen auswählen und der Experte kann nicht vollständig ausgeführt werden. Wenn Sie beispielsweise Ihrem Projekt eine Unit hinzufügen und dieser Unit Testfälle hinzufügen möchten, muss die Unit mindestens ein Klasse enthalten, damit dieser Experte erfolgreich ausgeführt werden kann.
Verfügbare Klassen und Methoden	Zeigt eine Baumhierarchie der für die aktuelle Unit verfügbaren Klassen und Methoden an. Sie können in der Baumhierarchie Elemente demarkieren. Standardmäßig sind alle Klassen und Methoden ausgewählt. Wenn Sie einzelne Methoden einer Klasse aus der Auswahl entfernen, ignoriert der Experte beim Erstellen der Testfälle diese Methoden. Wenn Sie eine Klasse ausschließen, ignoriert der Experte die gesamte Klasse und alle ihre Methoden, auch wenn Sie die Methoden nicht einzeln aus der Auswahl entfernt haben. Wenn Sie eine Klasse auswählen, aber keine ihrer Methoden markieren, erzeugt der Experte einen Testfall für die Klasse, generiert aber keine Testmethoden dafür.

Siehe auch

Überblick über das Testen von Units (siehe Seite 1425)

2.154 Testfall-Experte

Datei ▶ **Neu** ▶ **Weitere** ▶ **Unit-Test** ▶ **Testfall**

Verwenden Sie diese Seite des Experten, um Einzelheiten für den zu erstellenden Testfall anzugeben.

Element	Beschreibung
Testprojekt	Legt den Namen des Projekts fest. Enthält als Vorgabe den Namen des aktuellen Projekts.
Dateiname	Gibt den Dateinamen der Quelltextdatei an, die die Klassen und Methoden enthält, die getestet werden sollen. Enthält als Vorgabe den Namen der aktuellen Quelltextdatei.
Test-Framework	Gibt das Test-Framework an, das Sie verwenden möchten. RAD Studio erkennt automatisch den Typ des Projekts, das Sie testen, und legt dafür das korrekte Framework fest, das aktuell für die Quelltext-Personality unterstützt wird.
Basisklasse	Legt die Basisklasse fest, die durch den Test vererbt werden soll. Per Vorgabe wird der Testfall anhand der Basisklasse der aktiven Quelltextdatei erstellt. Optional.

Siehe auch

Überblick über das Testen von Units (siehe Seite 1425)

2.155 Testprojekt-Experte

Datei ▶ **Neu** ▶ **Weitere** ▶ **Unit-Tests** ▶ **Testprojekt**

Verwenden Sie diesen Experten, um eine Testfolge zu erstellen, die eine Reihe von Einzeltests enthält.

Element	Beschreibung
Projektname	Geben Sie den Namen des Projekts ein oder übernehmen Sie die Vorgabe. Die Vorgabe ist der Name des aktiven Projekts.
Speicherort	Geben Sie den Pfad für die Projektdatei an.
Personality	Wählen Sie eine Personality aus. Die Vorgabe wird auf der Basis des Typs des aktiven Projekts ermittelt.
Zur Projektgruppe hinzufügen	Wählen Sie aus, ob das neue Testprojekt der aktuellen Projektgruppe hinzugefügt werden soll. Per Vorgabe ist diese Option aktiviert. Wenn Sie ein Testprojekt erstellen, das in mehreren Projekten verwendet werden soll, deaktivieren Sie diese Option.

Siehe auch

Überblick über das Testen von Units (siehe Seite 1425)

2.156 Testprojekt-Experte

Datei ▶ **Neu** ▶ **Weitere** ▶ **Unit-Tests** ▶ **Testprojekt**

Verwenden Sie diese Seite des Experten, um das Framework und den Test-Runner für das Testprojekt festzulegen.

Element	Beschreibung
Test-Framework	Geben Sie das Framework an, das zum Erstellen des Testprojekts verwendet werden soll. Die Vorgabe ist für Delphi für .NET- und C# -Personalities NUnit ; für Delphi für Win32- und C++ kann nur DUnit verwendet werden. Für die C#-Personality wird nur NUnit unterstützt.
Test-Runner	Legen Sie den Test-Runner fest, der zur Ausführung des Testprojekts verwendet werden soll. Die Vorgabe ist GUI. Sie können auch Konsole auswählen. Die Testprojektassemblierung enthält dann einen Befehl zum Ausführen des Konsolentest-Runners aus dem Framework-Verzeichnis heraus.

Siehe auch

Überblick über das Testen von Units (siehe Seite 1425)

2.157 Erweiterte Datenbindung

Dieses Dialogfeld ist eine Komponente des Microsoft .NET Framework.

Verwenden Sie dieses Dialogfeld, um eine Eigenschaft an einen Wert von einem gültigen Datenprovider zu binden.

Weitere Informationen finden Sie in den Dokumentationsressourcen im Microsoft Developer Network (MSDN)

2.158 AutoFormat

Dieses Dialogfeld ist eine Komponente des Microsoft .NET Framework.

Verwenden Sie dieses Dialogfeld, um ein vordefiniertes Format mit Rahmen, Farben und Füllmustern für ein Tabellensteuerelement zu übernehmen.

Weitere Informationen finden Sie in den Dokumentationsressourcen im Microsoft Developer Network (MSDN)

2.159 Auflistungs-Editor

Dieses Dialogfeld ist eine Komponente des Microsoft .NET Framework.

Verwenden Sie dieses Dialogfeld, um einzelne Elemente einer Auflistung zu erstellen und zu bearbeiten.

Weitere Informationen finden Sie in den Dokumentationsressourcen im Microsoft Developer Network (MSDN)

2.160 Datenbindungen

Dieses Dialogfeld ist eine Komponente des Microsoft .NET Framework.

Verwenden Sie dieses Dialogfeld, um ein Datenelement zu binden und Formatierungen festzulegen.

Weitere Informationen finden Sie in den Dokumentationsressourcen im Microsoft Developer Network (MSDN)

2.161 Dynamische Eigenschaften

Verwenden Sie dieses Dialogfeld, um dynamische Eigenschaften festzulegen, die in einer Konfigurationsdatei geändert werden können, ohne dass die Anwendung neu kompiliert werden muss.

Element	Beschreibung
Eigenschaften	Markieren Sie die Eigenschaften, die Sie der Konfigurationsdatei hinzufügen möchten.
Schlüsselzuweisung	Optional. Überschreiben Sie den Standardschlüsselnamen für die Eigenschaft, der in der Konfigurationsdatei erscheint. Dadurch können Sie für die Eigenschaft einen aussagekräftigeren Schlüsselnamen vergeben.

Anmerkung: In einer Konfigurationsdatei gespeicherte Eigenschaftswerte sind nicht sicher. Vertrauliche Informationen, wie z.B. Passwörter, sollten nicht als dynamische Eigenschaften gespeichert werden.

Siehe auch

[Einführung in dynamische Eigenschaften](#)

Dynamische Eigenschaften festlegen (siehe Seite 74)

2.162 Eigenschaften

Dieses Dialogfeld ist eine Komponente des Microsoft .NET Framework.

Verwenden Sie dieses Dialogfeld zum Festlegen der Eigenschaften des aktuell ausgewählten Objekts.

Weitere Informationen finden Sie in den Dokumentationsressourcen im Microsoft Developer Network (MSDN)

2.163 <AliasDef>

Das Element <AttributeDef> legt die Speicherungsdetails für ein Attribut fest, das in dem Modell definiert ist.. Eine Klasse kann mehrere Aliasdefinitionen haben und unter bestimmten Umständen können diese Definitionen auf dieselbe Tabelle verweisen. Normalerweise definiert eine Klasse aber einen einzelnen Alias mit einer einzelnen Schlüsselimplementierung.

Attribut	Beschreibung
Name	Der Name kann willkürlich festgelegt werden. Er muss nur innerhalb der Klasse und ihrer Superklassen eindeutig sein. Die Elemente <AttributeDef> und <SingleLinkDef> referenzieren diesen Namen.
Tabelle	Dieses Attribut gibt eine durch Komma getrennte Liste der Tabellenspalten an, in die das Attribut aufgenommen werden soll.
ExtentRequiresDiscriminator	Dieses Attribut ist true, wenn in einer Tabelle Daten aus mehr als einer Klasse gespeichert werden. Um eine bestimmte Klasse vollständig (z.B. alle Instanzen) zu ermitteln, ist ein Typdiskriminatror erforderlich, der die Daten, die zu anderen Klassen in der Tabellen gehören, herausfiltert.
IsMainAlias	Wenn eine Klasse mehr als einen Alias definiert, kann einer davon als Hauptalias gekennzeichnet werden. Der Hauptalias sollte über eine Schlüsselimplementierung für einen Schlüssel verfügen, der mit dem Attribut "IsId" markiert ist. Der Hauptalias ist für andere Aliase bei der Ausführung von Zusammenführungsoperationen (Joins) das bevorzugte Ziel.
ContainsStartTime	Dieses Attribut wird nur von der ECO-Standardzuordnung OR verwendet. Es wird bei der Objektversionierung eingesetzt. Es legt fest, dass der Alias eine Spalte enthält, in der die "Startzeit" des Objekts festgehalten wird. Die Startzeit wird einem Objekt bei der Erstellung zugewiesen.
ContainsStopTime	Dieses Attribut wird nur von der ECO-Standardzuordnung OR verwendet.

Das Element <AliasDef> enthält die folgenden Unterknoten:

- <KeyImpl>
- <DiscriminatorImpl>

Siehe auch

Benutzerdefinierte ECO-OR-Zuordnungsdateien

<KeyImpl> (siehe Seite 472)

<DiscriminatorImpl> (siehe Seite 467)

2.164 <AttributeDef>

Das Element <ClassDef> definiert Zuordnungsinformationen für eine Klasse , die in dem Modell definiert ist.

Attribute	Beschreibung
Name	Der Wert dieses Attributs muss mit dem Namen des Attributs in dem Modell übereinstimmen. Das Attribut kann in der Klasse selbst oder in einer Vorfahrklasse festgelegt werden.
Alias	Dieses Attribut gibt an, welcher Alias beim Laden oder Speichern des Attributs verwendet wird. Wenn die Klasse nur einen Alias definiert und die globale Einstellung <i>ImplicitAliasInFeatures</i> true ist, kann dieses Attribut weggelassen werden.
Spalten	Dieses Attribut gibt eine durch Komma getrennte Liste der Tabellenspalten an, in die das Attribut aufgenommen werden soll. Wenn die Klasse nur einen Alias definiert und die globale Einstellung <i>ImplicitAliasInFeatures</i> true ist, kann dieses Attribut weggelassen werden.
AttributeMapper	Dieses Attribut sollte mit dem Namen der <i>PersistenceMapperDefinition</i> übereinstimmen, die in der <i>PersistenceMapper</i> -Komponente im EcoSpace definiert ist. Der <i>PersistenceMapper</i> hat eine Untereigenschaft namens <i>SqlDatabaseConfig.PersistenceMappers</i> . Diese Untereigenschaft ist eine Kollektion von Mapper-Komponenten, die darüber informiert sind, wie unterschiedliche Datentypen in den verschiedenen von ECO-Framework unterstützten Datenbanken gespeichert werden. Wenn dieses Attribut weggelassen wird oder leer ist, wird der Name des Attributs (z.B. Person.FirstName) verwendet. Wenn kein derartiger Mapper vorhanden ist, wird der .NET-Datentyp des Attributs (z.B. System.String) zum Suchen des Attribut-Mappers verwendet.
AllowNULL	Dieses Boolesche Attribut gibt an, ob die Tabellenspalte, in der das Attribut gespeichert ist, NULL als gültigen Wert akzeptieren soll. Dieses Attribut wird nur beim Erzeugen eines Datenbankschemas verwendet. Der Standardwert ist false .
Größe	Dieses Attribut legt die Anzahl der Zeichen fest, die ein String enthalten kann. Dieses Attribut wird nur beim Erzeugen eines Datenbankschemas verwendet. Der Standardwert ist 255 .

Siehe auch

Benutzerdefinierte ECO-OR-Zuordnungsdateien

2.165 <ClassDef>

Das Element <ClassDef> definiert Zuordnungsinformationen für eine Klasse, die in dem Modell definiert ist. Wenn die Klasse keine Persistenzmerkmale (Attribute oder Einzelbeziehungen) festlegt, kann die <ClassDef>-Spezifikation leer sein, sie muss aber vorhanden sein.

Attribute	Beschreibung
Name	Dieses Attribut muss mit dem Namen einer Klasse in dem Modell übereinstimmen.
SuperClass	Dieses Attribut muss mit dem Namen einer Superklasse der Klasse übereinstimmen. Diese Information muss in der Zuordnungsdatei doppelt vorhanden sein, weil das "alte" Modell nicht mehr zur Verfügung steht, wenn die Datenbankevolution durchgeführt wird.

Unterknoten des Elements <ClassDef>

- <AliasDef>
- <KeyDef>
- <DiscriminatorDef>
- <DiscriminatorValue>
- <AttributeDef>
- <SingleLinkDef>

Siehe auch

Benutzerdefinierte ECO-OR-Zuordnungsdateien

<AliasDef> (siehe Seite 460)

<KeyDef> (siehe Seite 471)

<DiscriminatorDef> (siehe Seite 466)

<DiscriminatorValue> (siehe Seite 468)

<AttributeDef> (siehe Seite 461)

<SingleLinkDef> (siehe Seite 473)

2.166 <Classes>

Das Element <Classes> ist der oberste Knoten, der alle <ClassDef>-Knoten in der OR-Zuordnungsdatei gruppiert.

Das Element <Classes> hat keine Attribute. Es enthält nur <ClassDef>-Unterknoten. Für jede persistente Klasse, für die Attribute oder Einzelbeziehungen in dem Modell definiert sind, muss ein <ClassDef>-Knoten vorhanden sein.

Siehe auch

[Benutzerdefinierte ECO-OR-Zuordnungsdateien](#)

2.167 <ConstantColumn>

Eine konstante Spalte ist eine Spalte, die für diesen Alias immer denselben Wert enthält. Dies entspricht etwa den Typdiskriminatorspalten, der Wert ist aber für den Alias, nicht für die Klasse (wie bei Typdiskriminatorspalten) konstant.

Attribute	Beschreibung
Name	Name der Tabellenspalte, die den konstanten Wert enthält.
Wert	Dieses Attribut legt den Wert fest, den die Spalte enthalten muss, damit sie für den Alias infrage kommt.
Signature	Dieses Attribut legt den .NET-Typ des Wertes (z.B. <i>System.String</i>) fest.

Siehe auch

Benutzerdefinierte ECO-OR-Zuordnungsdateien

2.168 <DiscriminatorColumn>

Das Element <DiscriminatorColumn> wird verwendet, wenn ein Typdiskriminator zum Suchen aller Objekte, die zu der Klasse gehören, erforderlich ist. Wenn zwei verschiedene Klassen in derselben Tabelle gespeichert sind, ist ein Typdiskriminator erforderlich, um nur die Objekte einer Klasse zu ermitteln.

Attribute	Beschreibung
Name	Der Name der Tabellenspalte, die für den Typdiskriminator verwendet wird. Dieser Name muss mit dem Spaltenamen eines <DiscriminatorImpl>-Elements in demselben Alias übereinstimmen.

Siehe auch

Benutzerdefinierte ECO-OR-Zuordnungsdateien

2.169 <DiscriminatorDef>

Das Element <DiscriminatorDef> wird zur Konfiguration der Typdiskriminatorspalten verwendet. Jede Klasse, die über einen Typdiskriminator verfügt, legt zunächst einen <DiscriminatorDef>-Knoten, der dem Diskriminator einen Namen gibt, und den Typ des Diskriminators fest. Dann kann der Diskriminator in einen Alias der Klasse und deren Subklassen mithilfe der Knoten <DiscriminatorImpl> "implementiert" werden. Die Werte, die in den Diskriminatorspalten verwendet werden sollen, müssen mit <DiscriminatorValue>-Knoten festgelegt werden.

Attribute	Beschreibung
Name	Dieses Attribut ist ein beliebiger Name, der in der Klasse und ihren Subklassen eindeutig sein muss. Er wird von den Elementen <DiscriminatorImpl> und <DiscriminatorValue> referenziert.
Signature	Dieses Attribut legt den .NET-Typ des Diskriminatorwertes (z.B. System.Int32 oder System.Char) fest.

Siehe auch

Benutzerdefinierte ECO-OR-Zuordnungsdateien

2.170 <DiscriminatorImpl>

Das Element <DiscriminatorImpl> implementiert den vom Knoten <DiscriminatorDef> definierten Typdiskriminator.

Attribute	Beschreibung
Name	Der von diesem Attribut festgelegte Name muss mit dem Namen des <DiscriminatorDef>-Knotens übereinstimmen, der in derselben oder einer Vorfahrklasse definiert ist.
Spalte	Dieses Attribut legt den Namen der Spalte fest, in der der Diskriminator gespeichert ist.

Siehe auch

Benutzerdefinierte ECO-OR-Zuordnungsdateien

2.171 <DiscriminatorValue>

Das Element <DiscriminatorValue> legt die Werte fest, die für eine bestimmte Klasse und für einen bestimmten Diskriminator verwendet werden sollen. Für jede <DiscriminatorDef>-Definition, die für die Klasse oder ihre Superklassen festgelegt ist, muss die Klasse einen <DiscriminatorValue>-Knoten festlegen (außer wenn eine Superklasse bereits einen Wert mit dem Attribut IsFinal definiert hat).

Attribute	Beschreibung
Name	Der Wert dieses Attributs muss einen Namen festlegen, der mit dem Namen eines <DiscriminatorDef>-Knotens übereinstimmt.
Wert	Dies ist die String-Repräsentation des Diskriminatorwertes, der für diese Klasse verwendet werden soll. Der Wert muss für die Diskriminatordefinition eindeutig sein.
IsFinal	Wenn der Diskriminatorwert für die Klasse und all ihre Subklassen übernommen werden soll, setzen Sie dieses Attribut auf true . Bei true können Subklassen keine neuen Werte für diesen Diskriminator festlegen.

Siehe auch

Benutzerdefinierte ECO-OR-Zuordnungsdateien

2.172 <Globals>

Der Knoten <Globals> ist der oberste Knoten in der OR-Zuordnungsdatei. Dieser Knoten legt globale Eigenschaften der OR-Zuordnungsdatei fest.

Attribute	Beschreibung
ImplicitAliasInFeatures	Wenn eine Klasse genau einen Alias festlegt, dann muss der Alias nicht für jedes Merkmal (Attribut oder Einzelbeziehung) angegeben werden. Der Standardwert für dieses Attribut ist false.
ImplicitColumnInFeatures	Wenn die Merkmale in allen <ClassDef>-Knoten die Eigenschaft columns nicht festlegen, wird der Name des Merkmals als Standardspaltenname verwendet. Der Standardwert für dieses Attribut ist false.

Siehe auch

Benutzerdefinierte ECO-OR-Zuordnungsdateien

2.173 <KeyColumn>

Das Element <KeyColumn> identifiziert die Tabellenspalte, in der der Schlüssel gespeichert ist. Wenn der Schlüssel zusammengesetzt ist, können mehrere <KeyColumn>-Knoten in einem <KeyImpl>-Element vorhanden sein.

Attribute	Beschreibung
Name	Der Name der Tabellenspalte.

Siehe auch

Benutzerdefinierte ECO-OR-Zuordnungsdateien

2.174 <KeyDef>

Jede Klasse, die instantiiert werden soll, muss mindestens ein <KeyDef>-Element besitzen. Das Element <KeyDef> ist aber nicht immer erforderlich. Beispielsweise haben abstrakte Klassen, die nicht instantiiert werden können, keine <KeyDef>-Elemente. Subklassen können das Element <KeyDef> von einer Superklasse erben, so dass das Element <KeyDef> nicht erforderlich ist.

Jede Klasse, die instantiiert werden soll, muss genau ein <KeyDef>-Element haben, dessen Attribut *IsId* auf **true** gesetzt ist. Dieses <KeyDef>-Element kann entweder zu einer Subklasse oder einer geerbten Klasse gehören.

Attribute	Beschreibung
Name	Der Name kann beliebig gewählt werden, muss aber in einer Klasse und ihren Subklassen eindeutig sein. Die Elemente <KeyImpl> und <SingleLinkDef> referenzieren diesen Namen.
Signature	Dieses Attribut ist eine durch Komma getrennte Liste der .NET-Typen für die Bestandteile des Schlüssels. Ein Beispiel ist ein 32-Bit-Integer-Schlüssel, System.Int32. Bei einem zusammengesetzten Schlüssel (z.B. Vor- und Nachname) lautet die Signatur "System.String, System.String".
IsId	Dieses Attribut gibt an, dass die Schlüsseldefinition als ID für die Klasse und ihre Subklassen verwendet werden soll. Jede persistente Klasse muss genau eine Schlüsseldefinition haben, die mit diesem Attribut gekennzeichnet ist (entweder von der Klasse selbst definiert oder von einer Superklasse).
KeyMapper	Dieses Attribut muss mit dem Namen übereinstimmen, der in der Konfiguration der im EcoSpace verwendeten Persistenz-Mapper-Komponente festgelegt ist. Die Untereigenschaft IdMappers des Persistenz-Mappers enthält eine Kollektion von KeyMapper-Klassen. Der Wert dieses Attributs sollte mit dem Namen eines Elements in dieser Kollektion übereinstimmen.

Es gibt vier vordefinierte Schlüsselzuordnungsklassen:

- Borland.Eco.Persistence.AttributeKeyMapper
- Borland.Eco.Persistence.AutoIncKeyMapper
- Borland.Eco.Persistence.GuidKeyMapper
- Borland.Eco.Persistence.DefaultIdMapper

Wenn das Attribut *KeyMapper* leer oder nicht angegeben ist, wird *DefaultIdMapper* verwendet.

Siehe auch

Benutzerdefinierte ECO-OR-Zuordnungsdateien

2.175 <KeyImpl>

Ein <KeyImpl>-Element legt die tatsächliche Verwendung einer Schlüsseldefinition fest. Jeder Alias muss mindestens ein <KeyImpl>-Element haben.

Attribute	Beschreibung
Name	Dieses Attribut muss mit dem Namen eines <KeyDef>-Knotens in der Klasse übereinstimmen, die den Alias festlegt (das könnte in einer Superklasse dieser Klasse vorkommen).
IsAutoInc	Für Schlüssel, die einen automatischen Inkrementierungsmechanismus verwenden, muss genau eine Schlüsselimplementierung vorhanden sein, die dieses Attribut als true definiert. Dies sollte der Schlüssel sein, der auf das Feld verweist, das den automatischen Inkrementierungswert bei Einfügeoperationen in die Datenbank mit INSERT erzeugt. Wenn der Schlüssel in mehreren Alias/Tabellen verwendet wird, erhalten die anderen Tabellen ihren Wert beim Erzeugen des Objekts in der Tabelle, die <i>IsAutoInc="true"</i> festlegt.

Das Element <KeyImpl> kann drei Unterknoten haben:

- <KeyColumn>
- <DiscriminatorColumn>
- <ConstantColumn>

Siehe auch

Benutzerdefinierte ECO-OR-Zuordnungsdateien

<KeyColumn> (siehe Seite 470)

<DiscriminatorColumn> (siehe Seite 465)

<ConstantColumn> (siehe Seite 464)

2.176 <SingleLinkDef>

Das Element <SingleLinkDef> legt eine Einzelbeziehungsassoziation fest, die in dem Modell definiert ist.

Attribute	Beschreibung
Name	<p>Der Wert dieses Attributs muss mit dem Namen des Assoziationsendes in dem Modell übereinstimmen.</p> <p>Beispielsweise hätte eine Klasse Person mit einer Einzelbeziehungsassoziation zu einer Klasse Building für Person das Element <SingleLinkDef> in dem Element <ClassDef>. Wenn das Assoziationsende home genannt würde, hätte das Attribut Name des <SingleLinkDef>-Elements den Wert "home".</p> <p>Das Attribut kann in der Klasse selbst oder in einer Vorfahrklasse festgelegt werden.</p>
Alias	Dieses Attribut gibt an, welcher Alias beim Laden oder Speichern des Attributs verwendet wird. Wenn die Klasse nur einen Alias definiert und die globale Einstellung <i>ImplicitAliasInFeatures</i> true ist, kann dieses Attribut weggelassen werden.
Spalten	Dieses Attribut gibt eine durch Komma getrennte Liste der Tabellenspalten an, in die das Attribut aufgenommen werden soll. Wenn die Klasse nur einen Alias definiert und die globale Einstellung <i>ImplicitAliasInFeatures</i> true ist, kann dieses Attribut weggelassen werden.
OrderColumn	Wenn die Beziehung sortiert ist, legt diese Attribut die Spalte fest, in der die Sortierinformationen enthalten sind. Diese Tabellenspalte kann auch als Bestandteil der ID verwendet und als Attribut bereitgestellt werden.
Key	Der Wert dieses Attributs muss die Form <code>ClassName.KeyName</code> haben. Zum Beispiel: <code>Key="Person.PersonId"</code> . Der Klassename muss mit dem Namen der Klasse übereinstimmen, auf die die Einzelbeziehung in dem Modell verweist, und der Schlüsselname muss mit dem Namen eines <KeyDef>-Knotens in dieser Klasse oder in einer ihrer Superklassen übereinstimmen.
IsConstrained	Dieses Boolesche Attribut gibt an, dass die Tabellenspalte, in der diese Beziehung gespeichert ist, eine referentielle Einschränkung hat. Damit wird die Reihenfolge gesteuert, in der Objekte in der Datenbank erzeugt und gelöscht werden, wenn eine Liste der Objekte aktualisiert wird.

Siehe auch

[Benutzerdefinierte ECO-OR-Zuordnungsdateien](#)

2.177 COM-Importe

Projekt ► Referenz hinzufügen

Verwenden Sie diese Seite, um dem aktuellen Projekt eine COM-Typbibliothek hinzuzufügen.

Das Dialogfeld Referenz hinzufügen kann auch in der Projektverwaltung aufgerufen werden, indem Sie den Ordner Referenzen mit der rechten Maustaste anklicken und Referenz hinzufügen wählen.

Element	Beschreibung
TypeLib-Name	Der Name der Typbibliothek.
TypeLib-Version	Die Version der Typbibliothek.
TypeLib-Pfad	Der Speicherort der Typbibliothek.
Referenz hinzufügen	Fügt die markierte Referenz der Liste Neue Referenzen hinzu.
Durchsuchen	Zeigt ein Dialogfeld an, in dem Sie zu einer Typbibliothek navigieren können.
Entfernen	Entfernt die aktuell in der Liste Neue Referenzen markierte Referenz aus der Liste.
OK	Wenn die Liste Neue Referenzen enthält, werden diese nach Anklicken von OK dem Projekt hinzugefügt.

Tip: Klicken Sie zum Sortieren der Anzeige eine beliebige Spaltenüberschrift an.

2.178 .NET-Assemblierungen

Projekt ► Referenz hinzufügen

Verwenden Sie diese Seite, um dem aktuellen Projekt eine .NET-Assemblierungsreferenz hinzuzufügen.

Das Dialogfeld Referenz hinzufügen kann auch in der Projektverwaltung aufgerufen werden, indem Sie den Ordner Referenzen mit der rechten Maustaste anklicken und Referenz hinzufügen wählen.

Element	Beschreibung
Name Assemblierung	der Der Name der Assemblierung.
Version	Die Version der Assemblierung.
Path	Der Speicherort der Assemblierung.
Referenz hinzufügen	Fügt die markierte Referenz der Liste Neue Referenzen hinzu.
Durchsuchen	Zeigt ein Dialogfeld an, in dem Sie zu einer Assemblierung navigieren können.
Entfernen	Entfernt die aktuell in der Liste Neue Referenzen markierte Referenz aus der Liste.
OK	Wenn die Liste Neue Referenzen Referenzen enthält, werden diese nach Anklicken von OK dem Projekt hinzugefügt.

Tip: Klicken Sie zum Sortieren der Anzeige eine beliebige Spaltenüberschrift an.

2.179 Projektreferenzen

Projekt>Referenz hinzufügen

Verwenden Sie diese Seite, um einem Projekt, das eine Assemblierung (.dll), wie z.B. eine Klassenbibliothek oder Steuerelementbibliothek, erzeugt, eine Referenz hinzuzufügen. Die Referenz wird dem aktuellen Projekt hinzugefügt.

Das Dialogfeld Referenz hinzufügen kann auch in der Projektverwaltung aufgerufen werden, indem Sie den Ordner Referenzen mit der rechten Maustaste anklicken und Referenz hinzufügen wählen.

Element	Beschreibung
Projektname	Der Name des Projekts, das eine Assemblierung erzeugt. Es werden nur Projekte aus der aktuellen Projektgruppe aufgeführt.
Projektverzeichnis	Der Speicherort des Projekts.
Referenz hinzufügen	Fügt die markierte Referenz der Liste Neue Referenzen hinzu.
Durchsuchen	Zeigt ein Dialogfeld an, in dem Sie zu einer Assemblierung navigieren können.
Entfernen	Entfernt die aktuell in der Liste Neue Referenzen markierte Referenz aus der Liste.
OK	Wenn die Liste Neue Referenzen enthält, werden diese nach Anklicken von OK dem Projekt hinzugefügt.

Tip: Klicken Sie zum Sortieren der Anzeige eine beliebige Spaltenüberschrift an.

2.180 Der Objektablage hinzufügen

Projekt>Der Objektablage hinzufügen

Verwenden Sie dieses Dialogfeld, um eine benutzerdefinierte Projekt-Template in der Objektablage zur Wiederverwendung in anderen Projekten zu speichern. Wenn Sie **Datei>Neu>Weitere** wählen, werden die gespeicherten Formulare und Templates im Dialogfeld Objektgalerie angezeigt.

Element	Beschreibung
Kategorie	Fügt die aktuellen Kategorienamen auf.
Neue Kategorie	Fügt der Objektablage einen Ordner mit einer neuen Kategorie hinzu.
Titel	Geben Sie einen Namen für die Template ein.
Beschreibung	Geben Sie eine Beschreibung für die Template ein. Die Beschreibung wird angezeigt, wenn Sie Datei>Neu>Weitere wählen, die Template in der Objektablage markieren, rechtsklicken und aus dem Kontextmenü Details anzeigen wählen.
Autor	Geben Sie den Namen des Autoren der Anwendung ein. Die Autorinformation wird nur angezeigt, wenn Sie Datei>Neu>Weitere wählen, die Template markieren, rechtsklicken und aus dem Kontextmenü Details anzeigen wählen.
Durchsuchen	Öffnet das Dialogfeld Symbol wählen, in dem Sie ein Symbol auswählen können, mit dem das Element in der Objektablage dargestellt wird. Sie können Bitmaps beliebiger Größen verwenden, die jedoch immer im Format 60 x 40 Pixel angezeigt werden.

Tip: Sie können den Pfad angeben, in dem nach der Datei BorlandStudioRepository.xml gesucht werden soll. Diese Datei beschreibt, wo die Objektablagen-Templates gespeichert sind. Wenn Sie den Pfad ändern möchten, wählen Sie **Tools>Optionen>Umgebungsoptionen** und geben den Pfad in das Textfeld Verzeichnis ein.

Nach dem Speichern eines Formulars oder Projekts als Template können Sie die Beschreibung bearbeiten, die Template ändern oder das Symbol ändern, indem Sie **Tools>Objektablage** wählen und die Schaltfläche Bearbeiten anklicken.

2.181 UDDI-Browser

Projekt ▶ Webreferenz hinzufügen

Verwenden Sie dieses Dialogfeld zum Suchen nach Services und Providern in den UDDI-Services-Sites mit WSDL-Services. Suchen Sie nach Namen oder durchsuchen Sie verfügbare Kategorisierungsschemata.

Element	Beschreibung
Microsoft Production	Ruft die Microsoft Production Site für vorhandene Web-Services auf.
Microsoft Test	Ruft die Microsoft Test Site für vorhandene Web-Services auf.
Neueste XMethods	Ruft die Homepage www.xmethods.com/ auf und stellt die neuesten Listings mit Web-Services bereit.
Vollständige XMethods	Ruft eine vollständige Liste mit allen verfügbaren Web-Services auf.
IBM Secure	Ruft den IBM UDDI Browser auf und ermöglicht die Suche nach UDDI Business-Registrierungen.
Ordnername Webreferenz	Zeigt den Namen der Webreferenz so an, wie er in dem aktiven WSDL-Dokument definiert ist.
Referenz hinzufügen	Fügt die im Eingabefeld Ordnername der Webreferenz angezeigte Referenz dem Projekt hinzu und erzeugt eine Proxy-Unit für die Webreferenz.

2.182 ASP-Deployment-Manager

Um das Dialogfeld des Deployment-Manager zu öffnen, wählen Sie die **Projektverwaltung** aus, klicken mit der rechten Maustaste auf den Knoten **Deployment** und wählen dann die Option **Neues ASP.NET-Deployment**. Mit diesem Dialogfeld können Sie automatisch die ASPX-, ASAX-, WEB.CONFIG-Dateien und die zugehörigen Assemblierungsdateien sammeln, die für das Deployment Ihres ASP.NET oder IntraWeb-Projekts erforderlich sind.

Seite "Deployment"

Verwenden Sie diese Seite, um die Speicherorte und den Status der Quell- und Zielfile für das ASP.NET-Deployment anzugeben.

Element	Beschreibung
Quellverzeichnis	Gibt das Verzeichnis an, in dem sich Ihr Projekt befindet.
Ziel	Aus dieser Dropdown-Liste können Sie ein Ziel für das Deployment auswählen. Das Ziel ist jenes Verzeichnis, in dem sich Ihre ausführbaren Dateien, DLL-, Auszeichnungs- und Konfigurationsdateien und alle anderen Dateien befinden, die für die Ausführung der Anwendung erforderlich sind. Das Zielverzeichnis kann ein Ordner oder eine FTP-Verzeichnis sein.
Quelldateien	Die meisten Dateien, die in Ihr Deployment-Verzeichnis gelegt werden müssen (z.B. Auszeichnungsdateien, ausführbare Dateien und Config-Dateien) werden im Diagramm der Quelldateien auf der linken Seite des Deployment-Managerfensters angezeigt.
Anmerkung:	Es kann erforderlich sein, den Knoten Referenzen für Ihr Projekt (in der Projektverwaltung) zu öffnen und die Option Lokal kopieren für manche DLL-Dateien auf True zu setzen, ehe sie sich in das Projekt einfügen lassen. (Angenommen, Sie arbeiten mit BDP so müssen Sie dies mit den Dateien Borland.Data.Common.dll und Borland.Data.Provider.dll ausführen.) Nachdem die Option Lokal kopieren gesetzt ist, compilieren Sie Ihr Projekt erneut. Diese zusätzlichen Dateien erscheinen dann ebenfalls in der Quell liste.

Um Datenbankspezifische Treiber hinzuzufügen, klicken Sie für Ihr Projekt mit der rechten Maustastse auf den Knoten Referenzen und wählen dann die Option Referenz hinzufügen. Wählen Sie den Treiber aus, der der verwendeten Datenbank entspricht. StatusIn der Spalte Status der Diagrammansicht auf der linken Seite des Deployment-Managerfensters wird der Status der Dateien angezeigt, die für das Deployment bereit stehen. Ehe Sie die Dateien weitergeben, sind sie mit dem Status **Nicht verbunden** gekennzeichnet. Sobald ein Zielort ausgewählt ist, wird sich der Status dieser Dateien in **Neu** ändern. ZieldateienGibt das Zielverzeichnis für Ihr Anwendungs-Deployment an. Nachdem Sie das Deployment-Managerfenster mit der rechten Maustaste angeklickt haben, können Sie aus dem Kontextmenü **Markierte Datei(en) nach Ziel kopieren** oder **Alle neuen oder geänderten Dateien nach Ziel kopieren** wählen. StatusIn der Spalte Status der Diagrammansicht auf der linken Seite des Deployment-Managerfensters wird der Status der Dateien angezeigt, die bereits weitergegeben wurden. Nachdem Sie Ihre Anwendung weitergegeben haben, werden die Dateien mit dem Status **Aktuell** aufgelistet.

Deployment-Listenfeld

Wenn Sie mit der rechten Maustaste klicken, während der Cursor im Deployment-Managerfenster steht, sehen Sie folgende Deployment-Optionen.

Element	Beschreibung
Aktualisieren	Zeigt die Listen nach den Änderungen erneut an.
Markierte Datei(en) nach Ziel kopieren	Kopiert alle angegebenen Quell-Dateien in das Zielverzeichnis.

Markierte Zielfile(l) löschen	Löscht die ausgewählten Dateien aus der Zielliste.
Name der Zielfile ändern	Ändert den Namen der Zielfile.
Alle neuen und geänderten Dateien nach Ziel kopieren	Kopiert alle angegebenen Quell-Dateien in das Zielverzeichnis.
Alle nicht im Projekt enthaltenen Zielfile löschen	Löscht alle Dateien aus der Zielliste, die nicht im Projekt enthalten sind.
Assemblierungsreferenzen anzeigen	Nach der Auswahl zeigt der Deployment-Manager alle Assemblierungen an, auf die im Projekt verwiesen wird. Die Systemassemblierungen werden zwar angezeigt, sind aber deaktiviert (grau). Dieses deaktivierten Assemblierungen können nicht weitergegeben werden.
Externe Dateien...	Dieser Befehl ermöglicht es, die externen Dateien auszuwählen, die weitergegeben werden sollen. Wenn Sie diese Option wählen, erscheint das Dialogfeld Externe Dateien mit einer Liste von Kontrollfeldern. Diese Liste enthält die BDP-Datenbankbibliotheken und die Datenbank-spezifischen Bibliotheken, da diese oft zusammen mit der ASP.NET-Anwendung weitergegeben werden müssen. Sie können mit dem Dialogfeld Datei öffnen weitere Dateien hinzufügen. Die Liste hat eine Spalte, in der das Zielunterverzeichnis für jede externe Datei angegeben wird. Sie können dieses Ziel bearbeiten. Aktivieren Sie die Kontrollfelder neben den Bibliotheken, die Sie dem Anwendungs-Deployment hinzufügen möchten und klicken Sie auf die Schaltfläche Hinzufügen.
Ignorierte Gruppen und Dateien anzeigen	Zeigt jene Gruppe und Dateien an, die Sie nicht einbeziehen möchten.
Gruppen ignorieren	Wählen Sie die Gruppen aus, die aus der Referenzliste herausgefiltert werden sollen.
Dateien ignorieren	Wählen Sie die Dateien aus, die aus der Referenzliste herausgefiltert werden sollen.
Protokollierung aktivieren	Erstellt ein Protokoll des Anwendungs-Deployment.
Protokoll anzeigen	Zeigt das Protokoll an, das beim Deployment des Projekts erstellt wurde.

2.183 Package ändern

Verwenden Sie dieses Dialogfeld, um Ihrem Package erforderliche Units hinzuzufügen. Dieses Dialogfeld wird angezeigt, wenn der Package-Editor versucht, ein Package zu compilieren und feststellt, dass dies nicht möglich ist oder dass das Package inkompatibel zu einem anderen Package ist, das aktuell in der IDE geladen ist. Das aktuelle Package verwendet nämlich eine oder mehrere Units, die sich in einem anderen Package befinden.

Element	Beschreibung
Details anzeigen	Zeigt eine Liste der Units an, die zum Erstellen des Package erforderlich sind.
OK	Fügt der <code>requires</code> -Klausel des aktuellen Package die fehlenden Packages hinzu.
Abbrechen	Verändert das aktuelle Package nicht.

Anmerkung: Wenn Sie auf Abbrechen klicken und die Änderungen nicht übernehmen, können beim Laden des Package Fehler auftreten.

2.184 Sprachen hinzufügen

[Projekt](#)▸[Sprachen](#)▸[Sprache hinzufügen](#)

Verwenden Sie diesen Experten zum Hinzufügen von einer oder mehreren Ressourcen-DLLs zu einem Projekt. Folgen Sie den Anweisungen auf den einzelnen Registerseiten des Experten.

Siehe auch

Anwendungen lokalisieren (☞ [siehe Seite 1437](#))

Sprachen zu einem Projekt hinzufügen (☞ [siehe Seite 86](#))

2.185 Sprachen entfernen

2

[Projekt](#)▶[Sprachen](#)▶[Sprache entfernen](#)

Verwenden Sie diesen Experten zum Entfernen von Sprachen aus Ihrem Projekt. Folgen Sie den Anweisungen auf den einzelnen Registerseiten des Experten.

Siehe auch

Anwendungen lokalisieren (↗ [siehe Seite 1437](#))

Sprachen zu einem Projekt hinzufügen (↗ [siehe Seite 86](#))

2.186 Aktive Sprache festlegen

[Projekt](#)▶[Sprachen](#)▶[Aktive Sprache festlegen](#)

Verwenden Sie diesen Experten zum Festlegen, welches Sprachmodul beim Ausführen der Anwendung in der IDE geladen werden soll. Bevor Sie die aktive Sprache ändern, müssen Sie sicherstellen, dass Sie die Satellite-Assemblierung für die gewünschte Sprache neu compiliert haben.

Markieren Sie die gewünschte Sprache und klicken Sie auf Fertig stellen.

Siehe auch

Anwendungen lokalisieren (☞ siehe Seite 1437)

Sprachen zu einem Projekt hinzufügen (☞ siehe Seite 86)

2.187 Neuer Kategorienname

Tools▶**Objektablage**▶**Schaltfläche Bearbeiten**▶**Schaltfläche Neue Kategorie**

Verwenden Sie dieses Dialogfeld, um einer neuen Kategorie in der Objektablage einen Namen zuzuweisen.

Element	Beschreibung
Kategorienname	Legt den Namen für die neue Kategorie fest, die der Objektablage hinzugefügt wird.

2.188 Entwurfs-Packages hinzufügen

Projekt▸**Optionen**▸**Packages**▸**Entwurfs-Packages Schalter Hinzufügen**

Verwenden Sie dieses Dialogfeld, um zu einem Entwurfs-Package zu navigieren und es der Liste Entwurfs-Packages hinzuzufügen.

2.189 Laufzeit-Package hinzufügen

Projekt▶ **Optionen**▶ **Packages**▶ **Ellipsen-Schaltfläche**▶ **Ellipsen-Schaltfläche**

Verwenden Sie dieses Dialogfeld zum Hinzufügen von Laufzeit-Packages zu der Liste Laufzeit-Packages.

Element	Beschreibung
Name	Geben Sie den Namen des Package an, das in die Liste Laufzeit-Packages aufgenommen werden soll, oder klicken Sie auf die Schaltfläche Durchsuchen, um im Dialogfeld Dateiname des Package nach dem Package zu suchen. Wenn das Package im Suchpfad enthalten ist, muss der Pfad nicht vollständig angegeben werden. (Wenn das Package-Verzeichnis nicht im Suchpfad steht, wird es am Ende angefügt.)
Suchpfad	Wenn Sie im Eingabefeld Name keinen vollständigen Verzeichnispfad angegeben haben, überprüfen Sie, ob das Verzeichnis mit dem Package in dieser Liste enthalten ist. Wenn Sie ein Verzeichnis in das Feld Suchpfad eingeben, ändern Sie damit auch den globalen Bibliothekssuchpfad. Zum Anzeigen einer sortierten, editierbaren Liste mit den Suchpfaden klicken Sie auf die Ellipsen-Schaltfläche. Sie können die Pfadliste auch in diesem Feld ändern.

2.190 Suchpfad für Symbole hinzufügen

Projekt▸**Optionen**▸**Symbole**▸**Neu**

In diesem Dialogfeld können Sie einen neuen Modulnamen und -pfad angeben, der der Liste der Symbole hinzugefügt werden soll. Der Modulname und -pfad wird der Liste auf der Seite der Symbole im Dialogfeld Projektoptionen hinzugefügt.

Element	Beschreibung
Modulname	Der Name des Moduls, das der Liste der Symbole hinzugefügt werden soll. Ist die Option Alle Symbole laden aktiv, werden alle Symbole geladen und die Liste der Module wird ignoriert. Ist die Option Alle Symbole laden nicht aktiv, können Sie die Namen der Module hinzufügen, nach denen der Debugger suchen soll, wenn die Symbole geladen werden. Beachten Sie, dass mit der Option Symbole für unspezifizierte Module laden lediglich jene aus dieser Liste nicht geladen werden. Der String wird als Moduldateiname interpretiert (z.B. foo.dll). Der String kann Platzhalter für die Angabe mehrerer Module enthalten. Sie können z.B. *core*.bpl angeben, um Module namens oldcore1.bpl , newcore2.bpl und so weiter zu kennzeichnen.
Pfad für Symbole	Eines oder mehrere Verzeichnisse mit dem Modul, das der Liste der Symbole hinzugefügt werden soll. Wenn Sie mehrere Verzeichnisse für ein Modul angeben, verwenden Sie einen Semikolon, um diese zu trennen. Klicken Sie auf die Schaltfläche ..., um das Dialogfeld Verzeichnisse anzuzeigen. Hieraus können Sie ein Verzeichnis auswählen.

Siehe auch

Überblick zum Debuggen (siehe Seite 1439)

Die Suchreihenfolge für Debug-Symbole festlegen (siehe Seite 36)

Symbole (siehe Seite 522)

2.191 Anwendung

Projekt ▶ Optionen ▶ Anwendung

Verwenden Sie dieses Dialogfeld zum Ändern des Namens und Typs der aktuellen Delphi für .NET-Anwendung.

Element	Beschreibung
Fehlersuche/Ausgabe	Zeigt die aktuell verwendete Gruppe mit Projektoptionen an. Die Einstellungen der Optionsgruppen für Fehlersuche und Ausgabe sind standardmäßig auf das Debuggen und Deployment einer Anwendung abgestimmt. Sie haben die Möglichkeit, benutzerdefinierte Optionsgruppen zu speichern und diese entsprechend der jeweiligen projektspezifischen Entwicklungsaktivität zu laden. Zur Erstellung einer benutzerdefinierten Optionsgruppe klicken Sie auf die Schaltfläche Speichern unter.
Speichern unter	Zeigt ein Dialogfeld für das Benennen und Speichern benutzerdefinierter Gruppen mit Projektoptionen an.
Löschen	Löscht die aktuelle Optionsgruppe. Dies ist nur bei benutzerdefinierten Optionsgruppen möglich. Die Standardoptionsgruppen für Fehlersuche und Ausgabe können nicht gelöscht werden.
Windows-Anwendung	Erstellt bei der nächsten Ausführung der Anwendung eine Windows-Anwendung. Der Quelltext wird dadurch nicht geändert.
Konsolenanwendung	Erstellt bei der nächsten Ausführung der Anwendung eine Konsolenanwendung. Der Quelltext wird dadurch nicht geändert.
Assemblierung	Erstellt bei der nächsten Ausführung der Anwendung eine Assemblierung. Der Quelltext wird dadurch nicht geändert.
Anwendungsname	Zeigt den aktuellen Namen der Anwendung an. Standardmäßig wird der Projektname als Anwendungsname verwendet. Sie können den Namen in diesem Feld bei Bedarf ändern.
Startobjekt	Gibt die Klasse an, in der die als Einsprungspunkt für das Programm verwendete <i>Main</i> -Methode enthalten ist. Dies ist vor allem dann von Nutzen, wenn das Programm mehrere Klassen mit einer <i>Main</i> -Methode enthält.
Standard-Namespace	Legt den Standard-Namespace für Elemente fest, die dem Projekt über das Dialogfeld Objektgalerie hinzugefügt werden. Per Vorgabe wird der Projektname (ohne Namenserweiterung) als Standard-Namespace verwendet.
Anwendungssymbol	Legt das Symbol (.ico) für die Ausgabedatei fest. Dieses Symbol wird im Windows-Explorer neben dem Namen der Ausgabedatei angezeigt.

2.192 Anwendung

Projekt ▶ Optionen ▶ Anwendung

Verwenden Sie dieses Dialogfeld zum Ändern des Namens und Typs der aktuellen Anwendung.

Anmerkung: Nicht alle im Folgenden beschriebenen Optionen stehen für alle Projekttypen zur Verfügung. Beispielsweise sind die Optionen für LIB-Attribute nicht für alle Projekte verfügbar.

Anwendungseinstellungen	Beschreibung
Titel	Tragen Sie hier den Titel der Anwendung ein, der unter dem Anwendungssymbol erscheinen soll, wenn die Anwendung als Symbol (minimiert) dargestellt wird. Maximal 255 Zeichen sind möglich.
Hilfdatei	Legen Sie den Speicherort der Hilfdatei für die angegebene Anwendung fest. Klicken Sie auf die Schaltfläche Durchsuchen..., um das Dialogfeld Hilfdatei der Anwendung anzuzeigen.
Symbol	Legt das Symbol (.ico) für die Ausgabedatei fest. Dieses Symbol wird im Windows-Explorer neben dem Namen der Ausgabedatei angezeigt. Klicken Sie auf die Schaltfläche Symbol laden..., um das Dialogfeld Programmsymbol anzuzeigen.
Laufzeit-Themes aktivieren	Legt fest, dass die Anwendung, die Sie entwickeln, Themes (für Windows Vista) verwenden soll. Der Standardwert lautet für vorhandene Projekte true und false für neue Projekte.

Ausgabeeinstellungen	Beschreibung
Erweiterung der Zieldatei	Legt die Erweiterung fest, die an die endgültige Version der ausführbaren Datei angehängt wird.

Bibliotheksnamenseinstellungen	Beschreibung
LIB-Präfix	Fügt ein angegebenes Präfix an den Namen der DLL- oder der Package-Ausgabedatei an.
LIB-Suffix	Fügt ein angegebenes Suffix vor der Dateiendung an den Namen der DLL- oder der Package-Ausgabedatei an.
LIB-Version	Fügt nach der Erweiterung eine zweite Erweiterung an den Namen der DLL- oder der Package-Ausgabedatei an. Beispiel: Wenn Sie 1.1.3 als Version für ein DLL namens WebApp festgelegt haben, erhielt die Ausgabedatei den Namen WebApp.dll.1.1.3.

Allgemeine Option	Beschreibung
Standard für neue Projekte	Speichert die aktuellen Einstellungen als Standardeinstellungen für neue Projekte.

2.193 Anwendung (Visual Basic)

Projekt ▶ Optionen ▶ Anwendung

Verwenden Sie dieses Dialogfeld zum Ändern des Namens und Typs der aktuellen Visual Basic-Anwendung.

Element	Beschreibung
Fehlersuche/Ausgabe	Zeigt die aktuell verwendete Gruppe mit Projektoptionen an. Die Einstellungen der Optionsgruppen für Fehlersuche und Ausgabe sind standardmäßig auf das Debuggen und Deployment einer Anwendung abgestimmt. Sie haben die Möglichkeit, benutzerdefinierte Optionsgruppen zu speichern und diese entsprechend der jeweiligen projektspezifischen Entwicklungsaktivität zu laden. Zur Erstellung einer benutzerdefinierten Optionsgruppe klicken Sie auf die Schaltfläche Speichern unter.
Speichern unter	Zeigt ein Dialogfeld für das Benennen und Speichern benutzerdefinierter Gruppen mit Projektoptionen an.
Löschen	Löscht die aktuelle Optionsgruppe. Dies ist nur bei benutzerdefinierten Optionsgruppen möglich. Die Standardoptionsgruppen für Fehlersuche und Ausgabe können nicht gelöscht werden.
Windows-Anwendung	Erstellt bei der nächsten Ausführung der Anwendung eine Windows-Anwendung. Der Quelltext wird dadurch nicht geändert.
Konsolenanwendung	Erstellt bei der nächsten Ausführung der Anwendung eine Konsolenanwendung. Der Quelltext wird dadurch nicht geändert.
Assemblierung	Erstellt bei der nächsten Ausführung der Anwendung eine Assemblierung. Der Quelltext wird dadurch nicht geändert.
Anwendungsname	Zeigt den aktuellen Namen der Anwendung an. Standardmäßig wird der Projektname als Anwendungsname verwendet. Sie können den Namen in diesem Feld bei Bedarf ändern.
Startobjekt	Legt die Klasse oder das Modul mit der <i>Sub Main</i> -Prozedur fest, die beim Start verwendet wird. Die Option entspricht der Visual Basic-Compileroption <i>/main</i> .
Standard-Namespace	Legt den Standard-Namespace für Elemente fest, die dem Projekt über das Dialogfeld Objektgalerie hinzugefügt werden. Per Vorgabe wird der Projektname (ohne Namenserweiterung) als Standard-Namespace verwendet.
Anwendungssymbol	Legt das Symbol (.ico) für die Ausgabedatei fest. Dieses Symbol wird im Windows-Explorer neben dem Namen der Ausgabedatei angezeigt. Die Option entspricht der Visual Basic-Compileroption <i>/win32icon</i> .

2.194 Optionsgruppe anwenden

[Projekt](#)▸[Optionen](#)▸[Laden...](#)

Verwenden Sie dieses Dialogfeld, um eine benannte Optionsgruppe für eine Projektkonfiguration anzuwenden.

Element	Beschreibung
Optionsgruppendatei	Navigiert zu der benannten Optionsgruppendatei. Klicken Sie auf [...], um ein Dialogfeld zur Dateiauswahl anzuzeigen.
Aktion	<p>Es gibt drei Möglichkeiten, die Werte zu übernehmen: Überschreiben, Ersetzen und Beibehalten. Überschreiben ersetzt die aktuelle Konfiguration vollständig durch die Werte aus der Optionsgruppe. Die Werte aus der Optionsgruppe überschreiben die aktuelle Konfiguration, und alle anderen Optionswerte werden auf ihre Standardwerte gesetzt. Sie erhalten hierbei genau dieselbe Konfiguration wie zu dem Zeitpunkt des ursprünglichen Speicherns der Optionsgruppe.</p> <p>Ersetzen schreibt alle Werte aus der Optionsgruppe in die aktuelle Konfiguration, ändert aber keine anderen Werte. Die Werte in der Optionsgruppe ersetzen die aktuellen Werte.</p> <p>Beibehalten schreibt nur die Werte aus der Optionsgruppe, die nicht schon in der aktiven Konfiguration gesetzt sind. Wenn die aktive Konfiguration Standardwerte für bestimmte Werte geändert hat, werden diese nicht verändert. Dadurch erhält die aktive Konfiguration Priorität.</p>

2.195 Cassini konfigurieren

Projekt ▶ Optionen ▶ Debugger ▶ ASP.NET

Verwenden Sie dieses Dialogfeld zum Festlegen des Pfades und der Port-Nummer für den Cassini Web Server. Das Dialogfeld wird nur angezeigt, wenn Sie Cassini nicht mit Tools ▶ Optionen ▶ ASP.NET-Optionen konfiguriert haben.

Element	Beschreibung
Path	Geben Sie den Pfad zu der ausführbaren Datei des Cassini Web Server an. Um nach dieser Datei zu suchen, klicken Sie die Ellipsen-Schaltfläche (...) an.
Port	Geben Sie den vom Cassini Web Server verwendeten TCP/IP-Port an.

Anmerkung: Wenn Sie keinen Zugriff auf den Cassini Web Server haben, können Sie ihn unter <http://www.asp.net/Projects/Cassini/Download> herunterladen.

2.196 Virtuelles Verzeichnis konfigurieren

Projekt ▶ Optionen ▶ ASP.NET

Verwenden Sie dieses Dialogfeld zum Festlegen und Konfigurieren des Servers und seines virtuellen Stammverzeichnisses für Ihre ASP.NET-Anwendung.

Element	Beschreibung
Speicherort	Legen Sie das Stammverzeichnis Ihrer ASP.NET-Anwendung fest.
Alias	Legen Sie den Namen für den Zugriff auf das virtuelle Verzeichnis fest.
Lesen	Ermöglicht das Lesen von Webserver-Scripts.
Scripts ausführen	Ermöglicht die Ausführung von ISAPI- und CGI-Anwendungen.
Execute	Ermöglicht das Speichern von Dateien in dem Verzeichnis.
Write	Markieren Sie dieses Kontrollkästchen, um Schreibberechtigungen zu aktivieren.
Durchsuchen	Ermöglicht den Zugriff auf Datei- und Verzeichnislisten.

2.197 ASP .NET

[Projekt](#) ▾ [Optionen](#) ▾ [Debugger](#) ▾ [ASP.NET](#)

Verwenden Sie dieses Dialogfeld zum Festlegen von Optionen für Ihre ASP.NET-Anwendungen.

Element	Beschreibung
Browser aufrufen	Markieren Sie das Kontrollkästchen, um automatisch einen Web-Browser aufzurufen, wenn der Befehl Start oder Ohne Debugger ausführen gewählt wird.
Startseite	Zeigt die .asp-Datei, die vom Web-Browser aufgerufen werden soll.
HTTP-Adresse	Geben Sie den Speicherort im Web für die angegebene .asp-Datei an.
Webserver als Host	Markieren Sie das Kontrollkästchen, um die Anwendung auf einem Webserver auszuführen, wenn der Befehl Start oder Ohne Debugger ausführen aufgerufen wird.
Server	Legen Sie den Standard-Webserver für neue Webanwendungen fest.
Virtuelles Verzeichnis	Legen Sie das Stammverzeichnis für die Anwendung fest.
Serveroptionen	Zeigt das Dialogfeld Virtuelles Verzeichnis konfigurieren an.
Standard	Speichert die aktuellen Projekteinstellungen als Standardeinstellungen für neue Projekte.

2.198 Manager für Build-Konfigurationen

Projekt ► Konfigurations-Manager

Verwenden Sie dieses Dialogfeld, um einem bestimmten Projekt oder Projekten eine benannte Build-Konfiguration zuzuweisen.

Element	Beschreibung
Name der Konfiguration	Zeigt die Namen der Build-Konfigurationen an. Die beiden Standardkonfigurationen (Debug und Ausgabe) sind zusammen mit den Build-Konfigurationen aufgelistet, die Sie im Dialogfeld Projekt > Optionen erstellt und benannt haben.
Verfügbare Projekte	Listet die Namen Ihrer Projekte, der den Projekten zugewiesenen aktiven Konfigurationen und den Pfad zu den jeweiligen Projekten auf. Um die in Name der Konfiguration aufgeführte Build-Konfiguration zuzuweisen, wählen Sie ein Projekt oder Projekte aus und klicken auf Übernehmen.

Tip: Verwenden Sie zum Erstellen einer benannten Build-Konfiguration das Dialogfeld [Projekt > Optionen](#). Optionen lassen sich auf mehreren Seiten dieses Dialogfeldes zu einer benannten Konfiguration hinzufügen. Die Registerkarten Compiler, Compiler-Meldungen, Linker und Verzeichnisse/Bedingungen enthalten das Feld Build-Konfiguration.

Siehe auch

Überblick zu MSBuild (siehe Seite 1461)

Überblick zu Build-Konfigurationen (siehe Seite 1463)

Erstellen von benannten Build-Konfigurationen für C++ (siehe Seite 6)

Erstellen von benannten Build-Konfigurationen für Delphi (siehe Seite 7)

Übernehmen der aktiven Build-Konfiguration (siehe Seite 2)

2.199 Build-Ereignisse

Projekt>Optionen>Build-Ereignisse

Mit dem Dialogfeld Build-Ereignisse lassen sich Pre-Build-, Pre-Link- und Post-Build-Ereignisse festlegen. Die Ergebnisse der von Ihnen in diesem Dialogfeld festgelegten Befehle werden im Ausgabebereich angezeigt. Zum Steuern der Ausgabeebene wählen Sie **Tools>Optionen>Umgebungsoptionen** und legen die Verbosity-Ebene fest.

Anmerkung: Manche Build-Optionen stehen für einige Projekttypen nicht zur Verfügung.

Element	Beschreibung
Pre-Build	Geben Sie die Befehle, die vor dem Build ausgeführt werden sollen, in den Bereich Befehle ein. Zum Anzeigen eines Dialogfeldes, das Sie bei der Eingabe der Befehlsliste unterstützt, klicken Sie auf Bearbeiten.
Pre-Link	Nur für C++ . Geben Sie die Befehle, die vor dem Linken ausgeführt werden sollen, in den Bereich Befehle ein. Zum Anzeigen eines Dialogfeldes, das Sie bei der Eingabe der Befehlsliste unterstützt, klicken Sie auf Bearbeiten.
Post-Build	Geben Sie die Befehle, die ausgeführt werden sollen, wenn der Build-Prozess beendet ist, in den Bereich Befehle ein. Zum Anzeigen eines Dialogfeldes, das Sie bei der Eingabe der Befehlsliste unterstützt, klicken Sie auf Bearbeiten.

Siehe auch

Pre-Build- oder Post-Build-Ereignis (Dialogfeld) (siehe Seite 520)

Erstellen von Build-Ereignissen (siehe Seite 5)

2.200 CodeGuard

Projekt>Optionen>CodeGuard

Verwenden Sie die Seite CodeGuard, um die CodeGuard-Meldungen zu aktivieren und den Level der CodeGuard-Abdeckung anzugeben.

Alle diese Optionen entsprechen einer Compiler-Option für [bcc32](#).

Anmerkung: Nachdem Sie die CodeGuard-Einstellungen geändert haben, sollten Sie das Projekt neu erstellen, um Linker-Fehler zu vermeiden.

Element	Beschreibung
CodeGuard-Überprüfung	Aktiviert die CodeGuard-Meldungen für ein Projekt.
Globale Stack-Elemente überprüfen	<p>Kommandozeilenoption: -vGd</p> <p>Erstellt die Daten und Stacklayout-Beschreibungen zum schnellen Nachschlagen durch CodeGuard. Anhand dieser Beschreibungen kann CodeGuard Datenverluste und ungültige Zeiger auf lokale, globale und statische Variablen protokollieren. Sie sollten diese Option immer verwenden.</p>
this-Zeiger bei Eintritt in Elementfunktion überwachen	<p>Kommandozeilenoption: -vGt</p> <p>Erzeugt spezielle Epiloge für Elementfunktionen, Konstruktoren und Destruktoren. CodeGuard überprüft den this-Zeiger beim Eintritt in alle Methoden im C++ Quelltext.</p> <p>Diese Option protokolliert Methodenaufrufe zu gelöschten oder ungültigen Objekten, selbst dann, wenn die Methoden selbst nicht auf this zugreifen.</p>
Zeigerzugriffe überprüfen	<p>Kommandozeilenoption: -vGc</p> <p>Erzeugt Aufrufe zum Überprüfen von Inline-Zeigerzugriffen im Quelltext. Diese Option schränkt die Ausführungs geschwindigkeit stark ein.</p>

Siehe auch

[Überblick zu CodeGuard](#)

[CodeGuard verwenden](#)

[CodeGuard-Konfiguration \(Dialogfeld\)](#) (siehe Seite 712)

2.201 Compiler

Projekt ▶ Optionen ▶ Compiler

Verwenden Sie dieses Dialogfeld, um die C#-Compileroptionen für das aktuelle Projekt festzulegen.

Element	Beschreibung
Debug/Release	Zeigt die aktuell verwendete Gruppe mit Projektoptionen an. Die Einstellungen der Optionsgruppen für Debug und Release sind standardmäßig auf das Debuggen und Deployment einer Anwendung abgestimmt. Sie haben die Möglichkeit, benutzerdefinierte Optionsgruppen zu speichern und diese entsprechend der jeweiligen projektspezifischen Entwicklungsaktivität zu laden. Zur Erstellung einer benutzerdefinierten Optionsgruppe klicken Sie auf die Schaltfläche Speichern unter.
Speichern unter	Zeigt ein Dialogfeld für das Benennen und Speichern benutzerdefinierter Gruppen mit Projektoptionen an.
Löschen	Löscht die aktuelle Optionsgruppe. Dies ist nur bei benutzerdefinierten Optionsgruppen möglich. Die Standardoptionsgruppen für Debug und Release können nicht gelöscht werden.
Debug-Informationen	Veranlasst den Compiler zur Erzeugung von Debug-Informationen, die bei der nächsten Ausführung der Anwendung in eine Programmdatenbankdatei (.pdb) eingefügt werden. Diese Option entspricht der Compileroption /debug.
Optimierung	Veranlasst den Compiler zur Durchführung von Optimierungen, durch die die Ausgabedatei kleiner, schneller und effizienter wird. Ist die Option aktiviert, führt auch die CLR (Common Language Runtime) zur Laufzeit eine Optimierung des Quelltextes durch. Diese Option entspricht der Compileroption /optimize.
XML-Dokumentation erzeugen	Verarbeitet Dokumentationskommentare im Quelltext und erzeugt in dem Verzeichnis mit der Projektdatei (.bdsproj für das BDS-Format oder .proj für das MSBuild-Format) eine XML-Datei namens <code>ProjectDoc.xml</code> . Zeilen, die mit /// beginnen und vor einem benutzerdefinierten Typ (wie einer Klasse, einem Delegaten oder einem Interface) oder vor einem Element (wie einem Feld, einem Ereignis, einer Eigenschaft, einer Methode) oder einer Namespace-Deklaration stehen, können als Kommentare verarbeitet und in die Datei eingefügt werden.. Wenn Inkrementelles Build aktiviert ist, wird die Option XML-Dokumentation erzeugen ignoriert. Diese Option entspricht der Compileroption /doc.
Überlaufprüfung erzeugen	Gibt an, ob eine arithmetische Integer-Anweisung, die sich nicht im Gültigkeitsbereich des Schlüsselworts checked oder unchecked befindet und einen Wert außerhalb des Bereichs für den Datentyp zurückgibt, eine Laufzeit-Exception auslöst. Diese Option entspricht der Compileroption /checked.
Unsicheren Code zulassen	Lässt die Compilierung von Quelltext zu, der das Schlüsselwort unsafe enthält. Diese Option entspricht der Compileroption /unsafe.
Warnungen wie Fehler behandeln	Behandelt alle Warnungen als Fehler. Meldungen, die normalerweise als Warnungen ausgegeben würden, werden als Fehler gemeldet, und der Compilierungsprozess wird angehalten (es werden keine Ausgabedateien erstellt). Diese Option entspricht der Compileroption /warnaserror.

Warnungsebene	Legt die Warnungsebene für den Compiler fest. 0 Unterdrückt alle Warnungen. 1 Zeigt schwer wiegende Warnungen an. 2 Zeigt Warnungen der Ebene 1 und bestimmte, weniger schwer wiegende Warnungen an (z.B. für verborgene Klassenelemente). 3 Zeigt Warnungen der Ebene 2 und bestimmte, weniger schwer wiegende Warnungen an (z.B. für Ausdrücke, die immer zu true oder false ausgewertet werden). 4 Zeigt Warnungen der Ebene 3 und informative Warnmeldungen an. Dies ist die vorgegebene Warnungsebene für die Befehlszeileneingabe. Diese Option entspricht der Compileroption /warn .
Basisadresse	Gibt die Basisadresse an, an der die DLL geladen werden soll. Die Standardbasisadresse für eine DLL wird von der .NET Framework-CLR (Common Language Runtime) festgelegt. Sie kann als Dezimal- oder Hexadezimalzahl angegeben werden. Diese Option entspricht der Compileroption /baseaddress .
Build Ausgabeverzeichnis	in Gibt das Ausgabeverzeichnis an, in dem das Projekt erzeugt wird. Diese Option hat keine entsprechende Compiler-Option und wird nur für importierte Visual Studio C#-Projekte zur Verfügung gestellt.
Inkrementelles Build	Compiliert nur Methoden, die seit der letzten Compilierung geändert wurden. Bei der ersten Verwendung dieser Option wird eine Datei (.incr) mit Compilerinformationen erzeugt. Die Datei wird bei allen nachfolgenden Compilierungen ergänzt. Wenn Inkrementelles Build aktiviert ist, wird die Option XML-Dokumentation erzeugen ignoriert. Diese Option entspricht der Compileroption /incremental .
Definitionen	Geben Sie Symbole ein, auf die in bedingten Compiler-Direktiven verwiesen wird. Mehrere Symbole müssen durch ein Semikolon voneinander getrennt werden.
Ausgabeverzeichnis	Gibt das Verzeichnis an, in dem der Compiler die ausführbare Datei speichert. Diese Option entspricht der Compileroption /out .

Tip: Wenn bei der Compilierung eines Projekts die Compileroptionen im Fenster Meldungen angezeigt werden sollen, wählen Sie **Tools**▶**Optionen**▶**Umgebungsoptionen** und aktivieren die Option Befehlszeile anzeigen. Bei der nächsten Compilierung eines Projekts werden der zur Compilierung verwendete Befehl und die Antwortdatei im Fenster Meldungen angezeigt. In der Antwortdatei sind die Compiler-Optionen und die zu compilierenden Dateien aufgeführt.

2.202 Compiler

Projekt ▶ Optionen ▶ Compiler

Verwenden Sie diese Seite, um die Compiler-Optionen für das aktuelle Projekt festzulegen.

Anmerkung: Nicht alle im Folgenden beschriebenen Optionen stehen für alle Projekttypen zur Verfügung.

Elemente der Gruppe "Code-Erzeugung"	Beschreibung
Build-Konfiguration	Zeigt den Namen der aktuellen Build-Konfiguration an, die den Optionen auf dieser Seite zugeordnet ist. Es gibt zwei Standard-Build-Konfigurationen: Debug und Ausgabe. Um weitere Build-Konfigurationen zu erstellen, geben Sie in dieses Feld einen Namen ein und klicken auf Speichern unter. Um die in diesem Feld angezeigte Build-Konfiguration zu löschen, klicken Sie auf Löschen.
Optimierung	Steuert die Codeoptimierung. Wenn diese Option aktiviert ist (entspricht {\$O+}), führt der Compiler eine Reihe von Codeoptimierungen aus, wie z.B. Variablen in CPU-Register platzieren, allgemeine Unterausdrücke entfernen und Induktionsvariablen erzeugen. Bei deaktivierter Option (entspricht {\$O-}) werden diese Optimierungen nicht ausgeführt. Außer in bestimmten Testsituationen sollte die Codeoptimierung immer aktiviert sein. Die Optimierungen des Delphi-Compilers führen zu keinerlei Änderungen der Funktionsweise des Programms. Mit anderen Worten: Der Compiler führt keine "unsicheren" Optimierungen durch, die die besondere Aufmerksamkeit des Programmierers erfordern. Mit dieser Option lassen sich Optimierungen nur für eine gesamte Prozedur oder Funktion ein- oder ausschalten. Dies ist nicht für bestimmte Zeilen innerhalb einer Routine möglich.
Stack-Frames	Nur für Delphi für Win32. Steuert die Erzeugung von Stack-Frames für Prozeduren und Funktionen. Wenn die Option aktiviert ist (entspricht {\$W+}), werden Stack-Frames für Prozeduren und Funktionen auch dann erzeugt, wenn sie nicht benötigt werden. Bei deaktivierter Option (entspricht {\$W-}), werden Stack-Frames nur generiert, wenn die Verwendung lokaler Variablen durch die Routine dies erforderlich macht. Normalerweise wird der Status {\$W+} nur für bestimmte Debugger-Tools benötigt, die eine Generierung von Stack-Frames für alle Prozeduren und Funktionen verlangen.

Pentium-sicheres FDIV	<p>Nur für Delphi für Win32. Steuert die Erzeugung von Fließkommacode, der vor der fehlerhaften FDIV-Anweisung schützt, die in bestimmten (frühen) Pentium-Prozessoren enthalten ist. Windows 95, Windows NT 3.51 und spätere Windows-BS-Versionen enthalten Code, der den Pentium-FDIV-Fehler korrigiert und das System schützt. Wenn die Option aktiviert ist (entspricht {\$U+}), werden alle Fließkommadivisionen mit einer Laufzeitbibliotheksroutine ausgeführt. Beim ersten Aufruf einer Fließkommadivision prüft die Routine, ob die FDIV-Anweisung des Prozessors korrekt arbeitet, und aktualisiert die in der Unit System deklarierte Variable <i>TestFDIV</i> entsprechend. Deren Wert bestimmt bei den folgenden Fließkommadivisionen, wie vorzugehen ist.</p> <p>-1 bedeutet, dass die FDIV-Anweisung getestet und als fehlerhaft erkannt wurde.</p> <p>0 bedeutet, dass die FDIV-Anweisung noch nicht geprüft wurde.</p> <p>1 bedeutet, dass die FDIV-Anweisung getestet und als korrekt erkannt wurde.</p> <p>Bei Pentium-Prozessoren, die keine fehlerhafte FDIV-Anweisung enthalten, bewirkt die Aktivierung dieser Option nur eine geringfügige Leistungsminderung. Bei fehlerhaften Pentium-Prozessoren können Fließkommadivisionen bei aktiverter Option bis zu dreimal länger dauern, liefern aber stets korrekte Ergebnisse. Bei deaktivierter Option (entspricht {\$U-}) werden Fließkommadivisionen durch Inline-FDIV-Anweisungen ausgeführt. Sie garantieren optimale Ausführungsgeschwindigkeiten und Codegrößen, können auf fehlerhaften Pentium-Prozessoren jedoch falsche Ergebnisse liefern. Sie sollten daher die Option nur deaktivieren, wenn Sie sicher sind, dass der Code auf einem fehlerfreien Pentium-Prozessor läuft.</p>
Record-Felder ausrichten	<p>Steuert die Ausrichtung von Feldern in Delphi-Record-Typen und Klassenstrukturen.</p> <p>Wenn Sie die Option 1 (entspricht {\$A1}) wählen oder die Option deaktivieren (entspricht {\$A-}), werden Felder nicht ausgerichtet. Alle Records und Klassenstrukturen werden gepackt.</p> <p>Wenn Sie die Option 2 (entspricht {\$A2}) wählen, werden die ohne den Bezeichner packed deklarierten Felder in Record-Typen und die Felder in Klassenstrukturen auf Wortgrenzen ausgerichtet.</p> <p>Wenn Sie die Option 4 (entspricht {\$A4}) wählen, werden die ohne den Bezeichner packed deklarierten Felder in Record-Typen und die Felder in Klassenstrukturen auf Doppelwortgrenzen ausgerichtet.</p> <p>Wenn Sie die Option 8 (entspricht {\$A8} oder {\$A+}) wählen, werden die ohne den Bezeichner packed deklarierten Felder in Record-Typen und die Felder in Klassenstrukturen auf Vierfachwortgrenzen ausgerichtet. Variablen und typisierte Konstanten werden unabhängig von der Direktive \$A immer für einen optimalen Zugriff ausgerichtet. Durch Setzen der Option auf 8 wird die Ausführung beschleunigt.</p>
Codeseite	Geben Sie die Codeseite für die Sprache Ihrer Anwendung ein. Die Codeseite wird als Dezimalzahl angegeben, die eine bestimmte Zeichencodierungstabelle bezeichnet. Es gibt Standardwerte für zahlreiche Sprachen.

Elemente der Gruppe "Syntaxoptionen"	Beschreibung
Strenge VAR-Strings-Prüfung	Diese Option (entspricht \$V) ist nur für Delphi-Quelltext von Bedeutung, in dem kurze Strings verwendet werden. Sie dient der Abwärtskompatibilität mit früheren Versionen von Delphi und CodeGear Pascal. Die Option steuert die Typenprüfung für kurze Strings, die als Variablenparameter übergeben werden. Wenn die Option aktiviert ist (entspricht {\$V+}), wird eine strenge Typenprüfung durchgeführt, d.h. formale und tatsächliche Parameter müssen denselben String-Typ haben. Bei deaktivierter Option (entspricht {\$V-}) kann eine Variable eines kurzen String-Typs selbst dann als Parameter verwendet werden, wenn die deklarierte Maximallänge nicht mit der des formalen Parameters übereinstimmt.

Vollständige Boolesche Auswertung	<p>Schaltet zwischen zwei unterschiedlichen Modellen der Delphi-Codegenerierung für die Booleschen Operatoren AND und OR um. Wenn die Option aktiviert ist (entspricht {\$B+}), erzeugt der Compiler Code für die vollständige Auswertung eines Booleschen Ausdrucks. Das bedeutet, dass jeder Operand eines Booleschen Ausdrucks, der mit den Operatoren AND und OR gebildet wird, garantiert ausgewertet wird, auch wenn das Ergebnis des gesamten Ausdrucks bereits feststeht. Bei deaktivierter Option (entspricht {\$B-}) generiert der Compiler Code für die Kurzschlussauswertung Boolescher Ausdrücke, d.h. die Auswertung wird beendet, sobald das Ergebnis des gesamten Ausdrucks feststeht (die Auswertung erfolgt immer von links nach rechts).</p>
Erweiterte Syntax	<p>Dient nur der Abwärtskompatibilität. Verwenden Sie diese Option (entspricht {\$X-}) nicht in Ihren Delphi-Anwendungen. Mit dieser Option lässt sich die erweiterte Syntax von Delphi aktivieren oder deaktivieren:</p> <p>Funktionsanweisungen Im Modus {\$X+} lassen sich Funktionsaufrufe als Prozederaufrufe verwenden, d.h. das Ergebnis eines Funktionsaufrufs kann ignoriert werden, anstatt an eine andere Funktion übergeben oder in einer Operation bzw. Zuweisung verwendet zu werden. Im Allgemeinen werden die von einer Funktion ausgeführten Berechnungen durch das Funktionsergebnis repräsentiert, das nicht ignoriert werden sollte. Manchmal führen Funktionen aber lediglich eine bestimmte Operation durch (z. B. einer globalen Variablen einen Wert zuweisen) und geben keinen Wert zurück, der weiterverwendet werden kann.</p> <p>Result-Variable Wenn die Option aktiviert ist (entspricht {\$X+}), kann die vordefinierte Variable <i>Result</i> für den Rückgabewert der Funktion verwendet werden.</p> <p>Nullterminierte Strings Wenn diese Option aktiviert ist, können Delphi-Strings nullbasierten Zeichen-Arrays (array[0..X] of Char) zugewiesen werden, die mit den PChar-Typen kompatibel sind.</p>
Typisierter @-Operator	<p>Steuert, welche Zeigertypen vom Operator @ generiert werden, und steuert deren Kompatibilität. Bei deaktivierter Option (entspricht {\$T-}) ist das Ergebnis des Operators @ immer ein Zeiger ohne Typ (Pointer), der mit allen übrigen Zeigertypen kompatibel ist. Wenn @ für einen Variablenverweis (entspricht {\$T+}) verwendet wird, ist das Ergebnis ein typisierter Zeiger, der nur mit <i>Pointer</i> und anderen Zeigern auf den Variablenotyp kompatibel ist. Bei deaktivierter Option sind festgelegte Zeigertypen (also nicht vom allgemeinen Typ <i>Pointer</i>) nicht kompatibel (auch wenn sie Zeiger auf denselben Typ sind). Bei aktiverter Option sind Zeiger auf denselben Typ kompatibel.</p>
Offene Parameter	<p>Ist nur für Quelltext von Bedeutung, der mit Huge-Strings compiliert wurde. Diese Option dient der Abwärtskompatibilität mit früheren Versionen von Delphi und CodeGear Pascal. Die Option (entspricht \$P) legt die Bedeutung von Variablenparametern fest, die mit dem Schlüsselwort string im Status {\$H-} deklariert wurden. Bei deaktivierter Option (entspricht {\$P-}) sind Variablenparameter, die mit dem Schlüsselwort string deklariert wurden, normale Variablenparameter. Dagegen werden sie im Status {\$P+} als OpenString-Parameter behandelt. Der Bezeichner OpenString kann unabhängig von der Einstellung der Direktive \$P immer zur Deklaration von OpenString-Parametern verwendet werden.</p>
Huge-Strings	<p>Nur für Delphi für Win32. Diese Option (entspricht die Direktive \$H) legt fest, welche Bedeutung das reservierte Wort string hat, wenn es ohne Zusatz in einer Typdeklaration steht. Der generische Typ <i>string</i> kann entweder einen langen, dynamisch zugewiesenen String (vom Typ <i>AnsiString</i>) oder einen kurzen, statisch zugewiesenen String (vom Typ <i>ShortString</i>) repräsentieren. Durch die Standardeinstellung wird der generische String-Typ als langer <i>AnsiString</i> definiert.</p> <p>Alle Objekte in der Komponentenbibliothek werden in diesem Status compiliert. Für neue Komponenten sollten ebenfalls lange Strings verwendet werden. Dasselbe gilt für Quelltext, der Daten aus String-Eigenschaften der Komponentenbibliothek übernimmt. Die deaktivierte Option (entspricht {\$H-}) ist sinnvoll, wenn auf Quelltext aus älteren Versionen von Delphi zugegriffen wird, in dem standardmäßig kurze Strings verwendet werden. Die Bedeutung von String-Typdefinitionen kann lokal überschrieben werden, um die Generierung kurzer Strings sicherzustellen. Außerdem können kurze String-Typen als <i>string[255]</i> oder <i>ShortString</i> deklariert werden. Diese Typen sind eindeutig und von der aktivierte Option unabhängig.</p>

Zuweisbare Konstanten	typisierte	<p>Legt fest, ob typisierte Konstanten geändert werden können. Wenn die Option aktiviert ist (entspricht {\$J+}), ist eine Änderung möglich. Typisierte Konstanten sind in diesem Fall mit initialisierten Variablen vergleichbar. Bei deaktivierter Option (entspricht {\$J-}) sind typisierte Konstanten tatsächlich konstant. Jeder Versuch, sie zu ändern, führt zu einer Fehlermeldung durch den Compiler. Als schreibbar werden typisierte Konstanten bezeichnet, die zur Laufzeit als Variablen verwendet und somit geändert werden können.</p> <p>Deshalb muss älterer Quelltext, der änderbare typisierte Konstanten enthält, mit aktiverter Option kompiliert werden, während für neue Anwendungen initialisierte Variablen verwendet werden sollten, und der Quelltext mit deaktivierter Option kompiliert werden sollte.</p>
-----------------------	------------	--

Elemente der Gruppe "Zielplattform"	Beschreibung (nur für .NET-Projekte)
Alle CPUs	Die ausführbare Datei kann mit allen CPUs ausgeführt werden.
x86	Die ausführbare Datei kann nur auf 32 Bit x86-CLRs (Common Language Runtime) ausgeführt werden.
x64	Die ausführbare Datei kann nur auf 64 Bit CLRs (Common Language Runtime) auf Computern ausgeführt werden, die das AMD64- oder EM64T-Anweisungsset unterstützen.

Elemente der Gruppe "Laufzeitfehler"	Beschreibung
Bereichsüberprüfung	Aktiviert oder deaktiviert die Generierung von Bereichsprüfungscode. Wenn die Option aktiviert ist (entspricht {\$R+}), werden alle Ausdrücke, die Arrays und Strings indizieren, dahingehend überprüft, ob sie sich innerhalb der festgelegten Grenzen befinden. Der gleichen Prüfung werden alle Zuweisungen an skalare Variablen und Telbereichsvariablen unterzogen. Das Fehlschlagen der Bereichsprüfung führt zu einer <i>ERangeError</i> -Exception (bzw. zum Programmabbruch, wenn die Exception-Behandlung nicht aktiviert ist). Die Aktivierung der Bereichsprüfung vergrößert und verlangsamt Ihr Programm.
E/A-Prüfung	Aktiviert oder deaktiviert die automatische Code-Generierung, die nach jedem Aufruf einer E/A-Prozedur das Ergebnis überprüft. Wenn eine E/A-Prozedur bei aktiverter Option ein Ergebnis ungleich 0 zurückgibt, führt dies zu einer <i>EInOutError</i> -Exception (bzw. zum Programmabbruch, wenn die Exception-Behandlung nicht aktiviert ist). Ist die Option deaktiviert, muss die E/A-Operation durch einen Aufruf von <i>IOResult</i> auf Fehler geprüft werden.
Überlaufprüfung	Steuert die Erzeugung von Code für die Überlaufprüfung. Wenn die Option aktiviert ist (entspricht {\$Q+}), werden bestimmte arithmetische Integer-Operationen (+, -, *, Abs, Sqr, Succ, Pred, Inc und Dec) auf einen Überlauf geprüft. Dazu wird ihnen zusätzlicher Code hinzugefügt, der sicherstellt, dass das Ergebnis innerhalb des unterstützten Bereichs liegt. Das Fehlschlagen der Überlaufprüfung führt zu einer <i>EIntOverflow</i> -Exception (bzw. zum Programmabbruch, wenn die Exception-Behandlung nicht aktiviert ist). Diese Option wird normalerweise zusammen mit der Option zur Bereichsprüfung (Schalter \$R) eingesetzt, die die Erzeugung von Bereichsprüfungscode aktiviert bzw. deaktiviert. Die Aktivierung der Überlaufsprüfung vergrößert und verlangsamt Ihr Programm.

Elemente der Gruppe "Debuggen"	Beschreibung
Debug-Informationen	Aktiviert oder deaktiviert die Generierung von Debug-Informationen. Diese Informationen beinhalten für jede Prozedur eine Tabelle mit Zeilennummern, in der Adressen des Objektcodes als Zeilennummern im Quelltext dargestellt werden. Bei Units werden die Debug-Informationen in der Unit-Datei gemeinsam mit dem Objektcode der Unit aufgezeichnet. Durch die Debug-Informationen erhöht sich die Größe der Unit-Datei. Das Compilieren von Programmen, die diese Unit verwenden, erfordert deshalb mehr Speicher. Die Größe und die Ausführungsgeschwindigkeit des ausführbaren Programms werden aber nicht nachteilig beeinflusst. Wenn ein Programm oder eine Unit mit dieser Option (entspricht {\$D+}) compiliert wird, können Sie das betreffende Modul mit dem integrierten Debugger in Einzelschritten testen und Haltepunkte setzen. Mit den Optionen Mit Debug-Infos und Map-Datei (auf der Seite Linker des Dialogfelds Projektoptionen) können nur vollständige Zeileninformationen für Module erzeugt werden, die mit aktiverter Option compiliert wurden. Diese Option wird normalerweise zusammen mit der Option Lokale Symbole (dem Schalter \$L) eingesetzt, der die Erzeugung von lokalen Symbolinformationen für das Debugging steuert.
Lokale Symbole	Aktiviert oder deaktiviert die Generierung von Symbolinformationen. Lokale Symbolinformationen sind die Namen und Typen aller lokalen Variablen und Konstanten eines Moduls, also die Symbole im implementation -Abschnitt des Moduls und in seinen Prozeduren und Funktionen. Bei Units werden die lokalen Symbolinformationen zusammen mit dem Objektcode in der Unit-Datei gespeichert. Lokale Symbolinformationen vergrößern eine Unit-Datei. Das Compilieren von Programmen, die diese Unit verwenden, erfordert also mehr Speicher. Die Größe und die Ausführungsgeschwindigkeit der Programme werden aber nicht nachteilig beeinflusst. Wenn ein Programm oder eine Unit mit dieser Option (entspricht {\$L+}) compiliert wird, können Sie die lokalen Variablen des Moduls im integrierten Debugger überprüfen und ändern. Außerdem können die Aufrufe von Prozeduren und Funktionen des Moduls über die Option Ansicht Aufruf-Stack überprüft werden. Mit den Optionen Mit TD32 Debug-Info und Map-Datei (auf der Seite Linker des Dialogfelds Projektoptionen) können nur Symbolinformationen für Module erzeugt werden, die mit aktiverter Option compiliert wurden. Diese Option wird normalerweise zusammen mit der Option Debug-Informationen eingesetzt, die die Erzeugung von Tabellen mit Zeilennummern zu Testzwecken steuert. Diese Option wird ignoriert, wenn der Compiler die Option Debug-Informationen deaktiviert hat.
Referenzinfo	Erzeugt für den Quelltext-Editor und die Projektverwaltung Informationen zu Symbolreferenzen. Entspricht {\$Y}. Wenn die Optionen Referenzinfo und Nur Definitionen aktiviert sind ({\$YD}), erzeugt der Compiler Informationen darüber, wo die Bezeichner definiert sind. Ist Referenzinfo aktiviert, aber Nur Definitionen deaktiviert ({\$Y+}), werden Informationen generiert, wo die Bezeichner definiert und verwendet werden. Diese Optionen sind nur wirksam, wenn Debug-Informationen und Lokale Symbole (siehe oben) aktiviert werden.
Nur Definitionen	Siehe Referenzinfo.
Assertion	Aktiviert oder deaktiviert die Generierung von Assert-Code in einer Quelltextdatei. Die Option ist standardmäßig aktiviert (entspricht {\$C+}). Gewöhnlich werden in der Auslieferungsversion eines Produkts zur Laufzeit keine Asserts verwendet. Entfernen Sie die Markierung von dieser Option, um Asserts zu deaktivieren.
Mit Debug-DCUIL/DCUs	Die Dateien DCUIL (.NET) oder DCU (Win32) enthalten Debug-Informationen und werden mit Stack-Frames compiliert. Ist diese Option aktiviert, fügt der Compiler automatisch den Debug-DCUIL/DCU-Pfad an den Anfang des im Feld Quellpfad für Debugger auf der Seite Verzeichnisse/Bedingungen angegebenen Suchpfades an.

Elemente der Gruppe "Dokumentation"	Beschreibung
XML-Dokumentation erzeugen	Erzeugt eine Datei, die die XML-Präsentation in Ihrem Projektverzeichnis enthält. Entspricht der Compiler-Option --doc.

Allgemeine Elemente	Beschreibung
Standard für neue Projekte	Speichert die aktuellen Einstellungen als Standardeinstellungen für neue Projekte.

2

Tip: Wenn bei der Compilierung eines Projekts die Compiler-Optionen im Fenster Meldungen angezeigt werden sollen, wählen Sie **Tools**▶**Optionen**▶**Umgebungsoptionen** und aktivieren die Option Befehlszeile anzeigen. Beim nächsten Compilieren eines Projekts werden der Befehl, der zur Compilierung des Projekts verwendet wurde, und die Antwortdatei im Fenster Meldungen angezeigt. In der Antwortdatei sind die Compiler-Optionen und die zu compilierenden Dateien aufgeführt.

Siehe auch

Compilieren (☞ siehe Seite 1351)

Erstellen von benannten Build-Konfigurationen für C++ (☞ siehe Seite 6)

Erstellen von benannten Build-Konfigurationen für Delphi (☞ siehe Seite 7)

2.203 Compiler-Meldungen

[Projekt](#) ▶ [Optionen](#) ▶ [Compiler-Meldungen](#)

Verwenden Sie diese Seite zur Steuerung der Informationen, die beim Compilieren vom Compiler bereitgestellt werden.

Element	Beschreibung
Build-Konfiguration	Zeigt den Namen der aktuellen Build-Konfiguration an, die den Optionen auf dieser Seite zugeordnet ist. Es gibt zwei Standard-Build-Konfigurationen: Debug und Release . Um weitere Build-Konfigurationen zu erstellen, geben Sie in dieses Feld einen Namen ein und klicken auf Speichern unter. Um die in diesem Feld angezeigte Build-Konfiguration zu löschen, klicken Sie auf Löschen.
Hinweise anzeigen	Ermöglicht das Anzeigen von Hinweisen beim Compilieren.
Warnungen anzeigen	Ermöglicht das Anzeigen von Warnungen beim Compilieren.
Warnungen	Ermöglicht die Auswahl der Warnungen, die beim Compilieren angezeigt werden sollen.
Standard für neue Projekte	Speichert die aktuellen Einstellungen als Standardeinstellungen für neue Projekte.

Siehe auch

Compiler-Fehler ( siehe Seite 1079)

2.204 Compiler (Visual Basic)

Projekt▸**Optionen**▸**Compiler**

Verwenden Sie dieses Dialogfeld, um die Visual Basic-Compileroptionen für das aktuelle Projekt festzulegen. Weitere Informationen zu Visual Basic-Compileroptionen finden Sie in der Online-Hilfe von .NET Framework SDK.

Element	Beschreibung
Fehlersuche/Ausgabe	Zeigt die aktuell verwendete Gruppe mit Projektoptionen an. Die Einstellungen der Optionsgruppen für Fehlersuche und Ausgabe sind standardmäßig auf das Debuggen und Deployment einer Anwendung abgestimmt. Sie haben die Möglichkeit, benutzerdefinierte Optionsgruppen zu speichern und diese entsprechend der jeweiligen projektspezifischen Entwicklungsaktivität zu laden. Zur Erstellung einer benutzerdefinierten Optionsgruppe klicken Sie auf die Schaltfläche Speichern unter.
Speichern unter	Zeigt ein Dialogfeld für das Benennen und Speichern benutzerdefinierter Gruppen mit Projektoptionen an.
Löschen	Löscht die aktuelle Optionsgruppe. Dies ist nur bei benutzerdefinierten Optionsgruppen möglich. Die Standardoptionsgruppen für Fehlersuche und Ausgabe können nicht gelöscht werden.
Debug-Informationen	Veranlasst den Compiler zur Erzeugung von Debug-Informationen, die bei der nächsten Ausführung der Anwendung in eine Programmdatenbankdatei (.pdb) eingefügt werden. Diese Option entspricht der Compileroption <code>/debug</code> .
Optimierung	Veranlasst den Compiler zur Durchführung von Optimierungen, durch die die Ausgabedatei kleiner, schneller und effizienter wird. Ist die Option aktiviert, führt auch die CLR (Common Language Runtime) zur Laufzeit eine Optimierung des Quelltextes durch. Diese Option entspricht der Compileroption <code>/optimize</code> .
Warnungen wie Fehler behandeln	Behandelt alle Warnungen als Fehler. Meldungen, die normalerweise als Warnungen ausgegeben würden, werden als Fehler gemeldet, und der Compilierungsprozess wird angehalten (es werden keine Ausgabedateien erstellt). Diese Option entspricht der Compileroption <code>/warnaserror</code> .
Warnungen anzeigen	Erzeugt Compiler-Warnungen. Diese Option entspricht der Compileroption <code>/nowarn</code> .
Definitionen	Geben Sie Symbole ein, auf die in bedingten Compiler-Direktiven verwiesen wird. Mehrere Symbole müssen durch ein Semikolon voneinander getrennt werden.
Ausgabeverzeichnis	Gibt das Verzeichnis an, in dem der Compiler die ausführbare Datei speichert. Diese Option entspricht der Compileroption <code>/out</code> .

Tip: Wenn bei der Compilierung eines Projekts die Compileroptionen im Fenster Meldungen angezeigt werden sollen, wählen Sie **Tools**▸**Optionen**▸**Umgebungsoptionen** und aktivieren die Option Befehlszeile anzeigen. Bei der nächsten Compilierung eines Projekts werden der zur Compilierung verwendete Befehl und die Antwortdatei im Fenster Meldungen angezeigt. In der Antwortdatei sind die Compiler-Optionen und die zu compilierenden Dateien aufgeführt.

2.205 Komponente

Projekt▶ **Optionen**▶ **Packages**▶ **Schalter Komponenten**

Verwenden Sie dieses Dialogfeld, um die Komponenten des ausgewählten Package zusammen mit den Symbolen anzuzeigen, die die Komponenten in der Tool-Palette repräsentieren, wenn das Package installiert ist.

2.206 Debugger

Projekt>Optionen>Debugger

Verwenden Sie dieses Dialogfeld zur Übergabe von Befehlszeilenparametern an die Anwendung, zur Festlegung einer Host-Anwendung für den Test einer DLL oder zum Laden einer ausführbaren Datei in den Debugger. Dieses Dialogfeld lässt sich auch über **Start>Parameter** aufrufen.

Element	Beschreibung
Host-Anwendung	Legt den Pfad zu einer ausführbaren Datei fest. (Klicken Sie auf Durchsuchen, um ein Dialogfeld zur Dateiauswahl aufzurufen.) Wenn es sich beim aktuellen Projekt um eine DLL oder ein Package handelt, können Sie hier eine Host-Anwendung für den Aufruf angeben. Hier kann auch der Name einer ausführbaren Datei angegeben werden, die im Debugger ausgeführt werden soll. Wenn ein geöffnetes Projekt ausgeführt werden soll, ist für das Feld Host-Anwendung keine Eingabe erforderlich.
Parameter	Legt die Befehlszeilenparameter fest, die an Ihre Anwendung werden sollen. Geben Sie hier die Befehlszeilenargumente ein, die beim Starten an Ihre Anwendung bzw. an die Host-Anwendung übergeben werden sollen. Mit der Pfeil-Schaltfläche können Sie eine Liste der zuletzt verwendeten Parameter öffnen.
Arbeitsverzeichnis	Legt den Namen des Debug-Verzeichnis fest. Standardmäßig wird das Verzeichnis mit der ausführbaren Datei der Anwendung verwendet.
Quellpfad	<p>Gibt die Verzeichnisse an, die die Quelltextdateien enthalten. Der Debugger verwendet standardmäßig die Pfadeinstellungen für den Compiler. Wenn die Verzeichnisstruktur seit der letzten Compilierung geändert wurde oder der Debugger keine Quelltextdatei finden kann, können Sie mit diesem Pfad die Quelltextdatei in die Debugging-Sitzung einschließen. Klicken Sie auf die Ellipsen-Schaltfläche, um ein Dialogfeld einzublenden, in dem Sie eine sortierte Liste mit Verzeichnisquellpfaden bearbeiten können. Die zusätzlichen Verzeichnisse werden in der folgenden Reihenfolge durchsucht:</p> <ol style="list-style-type: none"> 1. Pfad für Debugger (diese Option). 2. Der für Delphi für Win32 oder .NET oder für C++ festgelegte Suchpfad: <ul style="list-style-type: none"> — Für Delphi für Win32 auf der Seite Tools>Optionen>Umgebungsoptionen>Delphi-Optionen>Bibliothek — Win32. — Für Delphi.NET auf der Seite Tools>Optionen>Umgebungsoptionen>Delphi-Optionen>Bibliothek — NET. — For C++ auf der Seite Tools>Optionen>Debugger-Optionen>Borland-Debugger. 3. Pfad für Debugger, der auf der Seite Tools>Optionen>Delphi-Optionen>Borland-Debugger angegeben wurde. <p>Hinweis: Der Suchpfad wird für C# und Visual Basic nicht verwendet.</p>
Standard für neue Projekte	Speichert die aktuellen Einstellungen als Standardeinstellungen für neue Projekte.

2.207 Beschreibung

Projekt ▶ Optionen ▶ Beschreibung

In der Registerkarte Beschreibung hinterlegen Sie eine Beschreibung des Package mit Angaben zum Verwendungszweck und des Compilierungsmodus.

Anmerkung: Diese Seite erscheint nur, wenn Sie ein Package entwickeln.

Option	Beschreibung
Beschreibung	Beschreibung, die beim Installieren des Package angezeigt wird.

Verwenden für	Beschreibung
Verwenden für	Gibt die Verwendung eines Package an.
Entwurf	Das Package lässt sich in der Tool-Palette installieren.
Laufzeit	Das Package kann mit einer Anwendung weitergegeben werden.
Entwurf und Laufzeit	Das Package ist installierbar und für das Deployment bereit.

Package-Name	Beschreibung
LIB-Präfix	Fügt ein angegebenes Präfix an den Namen der Ausgabedatei an.
LIB-Version	Fügt dem Ausgabedateinamen eine zweite Dateinamenserweiterung nach der Erweiterung <code>.bpl</code> hinzu. Durch die Angabe <code>2.1.3</code> für <code>Package1</code> wird beispielsweise der Name <code>Package1.bpl.2.1.3</code> generiert.
LIB-Suffix	Fügt ein angegebenes Suffix vor der Dateiendung an den Namen der Ausgabedatei an. Durch die Angabe <code>-2.1.3</code> für <code>Package1</code> wird beispielsweise der Name <code>Package1-2.1.3.bpl</code> generiert.

Allgemeine Option	Beschreibung
Standard für neue Projekte	Speichert die aktuellen Einstellungen als Standardeinstellungen für neue Projekte.

2.208 Verzeichnisse/Bedingungen

Projekt>Optionen>Verzeichnisse/Bedingungen

Verwenden Sie diese Seite zum Festlegen der Pfade für Verzeichnisse und bedingte Definitionen. Sie können auch den Namespace setzen, um die `uses`-Klausel zu vereinfachen.

Element	Beschreibung
Build-Konfiguration	Zeigt den Namen der aktuellen Build-Konfiguration an, die den Optionen auf dieser Seite zugeordnet ist. Es gibt zwei Standard-Build-Konfigurationen: Debug und Release. Um weitere Build-Konfigurationen zu erstellen, geben Sie in dieses Feld einen Namen ein und klicken auf Speichern unter. Um die in diesem Feld angezeigte Build-Konfiguration zu löschen, klicken Sie auf Löschen.
Ausgabeverzeichnis	Gibt das Verzeichnis an, in dem der Compiler die ausführbare Datei speichert.
Ausgabe für Units	Legt ein separates Verzeichnis für die <code>.dcu</code> - (Win32) oder <code>.dcu1</code> - (.NET) Dateien fest.
Suchpfad	Legt den Pfad für die Quelltextdateien fest. Nur die Dateien in den Verzeichnissen, die hier oder im Suchpfad für Bibliotheken angegeben sind, werden vom Compiler berücksichtigt. Bei den anderen Dateien erhalten Sie einen entsprechenden Compiler-Fehler. Sie müssen den vollständigen Pfad angeben. Trennen Sie mehrere Verzeichnisnamen jeweils durch einen Doppelpunkt. Leerzeichen vor und nach dem Doppelpunkt sind erlaubt, aber nicht erforderlich. Sie können relative und absolute Pfadnamen angeben. Wenn Sie die Option Mit Debug-DCUILs auf der Registerkarte Projekt>Optionen>Compiler aktivieren, wird der mit Debug-DCUIL-Pfad (Tools>Debugger-Optionen>Borland .NET-Debugger) angegebene Pfad am Anfang des Suchpfads eingefügt.
Package-Ausgabeverzeichnis	Legt das Verzeichnis fest, in das die erstellten Package-Dateien geschrieben werden.
DCP/DCPIL-Ausgabeverzeichnis	Legt das Verzeichnis fest, in das die <code>.dcp</code> - (Win32) oder <code>.dcpl</code> - (.NET) Dateien während der Compilierung geschrieben werden. Wenn Sie hier keinen Wert angeben, wird das globale DCP/DCPIL-Ausgabeverzeichnis aus der Seite Tools>Optionen>Umgebungsoptionen>Delphi-Optionen>Bibliothek verwendet.
Bedingungen	Legen Sie Symbole fest, auf die in bedingten Compiler-Direktiven verwiesen wird. Mehrere Definitionen müssen durch ein Semikolon voneinander getrennt werden.
Unit-Aliase	Dient der Abwärtskompatibilität. Geben Sie hier Alias-Namen für Units an, die in neueren Versionen einen anderen Namen haben oder zu einer Unit zusammengefasst sind. Das Format lautet <code><AlteUnit>=<NeueUnit></code> (z.B. <code>Forms=Xforms</code>). Mehrere Aliase müssen durch ein Semikolon voneinander getrennt werden. Der voreingestellte Wert für Delphi ist: <code>WinTypes=Windows;WinProcs=Windows.Default</code> .
Namespace-Präfixe	Legt die Präfixe für Namespaces fest, um das Erstellen von Kurzversionen von Namespaces in den <code>uses</code> -Klauseln der Quelltextdatei zu ermöglichen. Z.B. können Sie anstatt <code>Borland.Vcl.DB</code> zu schreiben, das Namespace-Präfix <code>Borland.Vcl</code> festlegen. In der <code>uses</code> -Klausel können Sie dann <code>uses DB;.</code> verwenden
Standard-Namespace	Gibt den Standard-Namespace für alle Units des Projekts fest.
Standard für neue Projekte	Speichert die aktuellen Einstellungen als Standardeinstellungen für neue Projekte.

2.209 Debugger-Umgebung

Projekt ▶ Optionen ▶ Debugger ▶ Umgebung

Verwenden Sie diese Seite zur Angabe der Umgebungsvariablen, die während des Debuggens an die Anwendung übergeben werden. Dieses Dialogfeld lässt sich auch über **Start ▶ Parameter** aufrufen.

Element	Beschreibung
Systemvariablen	Zeigt eine Liste aller Umgebungsvariablen und ihrer Werte an, die auf Systemebene definiert sind. Eine Systemvariable kann nicht gelöscht, aber überschrieben werden.
Variable überschreiben	Öffnet das Dialogfeld Systemvariable überschreiben, in dem Systemvariablen geändert werden können, um neue vom Anwender überschriebene Variablen zu erstellen. Diese Schaltfläche ist deaktiviert, bis Sie eine Variable in der Liste Systemvariablen auswählen.
Vom Anwender überschrieben	Zeigt alle Anwendervariablen mit den zugehörigen Werten an. Anwendervariablen haben Vorrang vor Systemvariablen und verlieren ihre Gültigkeit erst, wenn sie gelöscht werden.
Neu	Öffnet das Dialogfeld Neue Anwendervariable, in dem Sie eine neue Anwendervariable aus einer Systemvariable erstellen können.
Bearbeiten	Öffnet das Dialogfeld Anwendervariable bearbeiten, in dem Sie die Anwendervariable ändern können, die in der Liste Vom Anwender überschrieben ausgewählt ist.
Löschen	Löscht die Anwendervariable, die in der Liste Vom Anwender überschrieben ausgewählt ist.
Mit Systemvariablen	Übergibt die Systemumgebungsvariablen an die Anwendung, für die der Debug-Vorgang ausgeführt wird. Wenn diese Option deaktiviert ist, werden nur die vom Anwender überschriebenen Variablen an die Anwendung übergeben.
Standard für neue Projekte	Speichert die aktuellen Einstellungen als Standardeinstellungen für neue Projekte.

2.210 Formulare

Projekt▶**Optionen**▶**Formulare**. Verwenden Sie diese Seite, um das Hauptformular für das aktuelle VCL-Formular-Projekt auszuwählen und festzulegen, welches der verfügbaren Formulare automatisch beim Start der Anwendung erstellt werden soll.

Element	Beschreibung
Hauptformular	Legt fest, welches Formular beim Start der Anwendung angezeigt wird. Wählen Sie aus der Dropdown-Liste das Hauptformular für das Projekt aus. Dieses Formular steht dann in der Liste Automatisch erzeugen an erster Stelle.
Automatisch erzeugen	Enthält die Formulare, die automatisch in den Startquelltext der Projektdatei eingefügt und zur Laufzeit erzeugt werden. Beim Aufrufen der Anwendung werden diese Formulare automatisch erzeugt und angezeigt. Die Reihenfolge, in der dies geschieht können Sie in der Liste per Drag&Drop anpassen. Wenn mehrere Formulare markiert werden sollen, halten Sie die Taste Umschalt gedrückt, während Sie die Formularnamen auswählen.
Verfügbare Formulare	Enthält die Formulare, die zwar von der Anwendung benötigt, aber nicht automatisch erzeugt werden. Wenn eine Instanz dieser Formulare erstellt werden soll, müssen Sie den Konstruktor Create der Formularklasse explizit aufrufen.
Pfeilschalter	Verwenden Sie die Pfeilschaltflächen, um Dateien von einer Liste in die andere zu verlagern.
Standard für neue Projekte	Speichert die aktuellen Einstellungen als Standardeinstellungen für neue Projekte.

2.211 Linker

Projekt>Optionen>Linker

Verwenden Sie diese Seite zum Festlegen von Linker-Optionen für Ihre Anwendung.

Anmerkung: Nicht alle im Folgenden beschriebenen Optionen stehen für alle Projekttypen zur Verfügung.

Elemente der Gruppe "Map-Datei"	Beschreibung
Build-Konfiguration	Zeigt den Namen der aktuellen Build-Konfiguration an, die den Optionen auf dieser Seite zugeordnet ist. Es gibt zwei Standard-Build-Konfigurationen: Debug und Ausgabe. Um weitere Build-Konfigurationen zu erstellen, geben Sie in dieses Feld einen Namen ein und klicken auf Speichern unter. Um die in diesem Feld angezeigte Build-Konfiguration zu löschen, klicken Sie auf Löschen.
Aus	Der Linker erzeugt keine Map-Datei.
Segmente	Der Linker generiert eine Map-Datei mit folgendem Inhalt: Liste der Segmente, Programmstartadresse und eventuelle Warn- und Fehlermeldungen, die während des Linkens erzeugt wurden.
Publics	Der Linker generiert eine Map-Datei mit folgendem Inhalt: Liste der Segmente, Programmstartadresse und eventuelle Warn- und Fehlermeldungen, die während des Linkens erzeugt wurden. Außerdem wird eine alphabetisch sortierte Liste der public-Symbole generiert.
Detailliert	Der Linker generiert eine Map-Datei mit folgendem Inhalt: Liste der Segmente, Programmstartadresse und eventuelle Warn- und Fehlermeldungen, die während des Linkens erzeugt wurden, eine alphabetisch sortierte Liste der public-Symbole und zusätzlich eine detaillierte Liste der Segmentzuordnungen. Diese umfasst Segmentadressen, Längen in Byte, Segmentnamen, Gruppe und Modulinformationen.
Typbibliothek auto-registrieren	Nur für Projekte mit einer Typbibliothek verfügbar. Wählen Sie die Schaltfläche Registrieren, wird die Typbibliothek in die Windows-Registrierung eingetragen.
Importassemblierung erzeugen	Nur für Projekte mit einer Typbibliothek verfügbar. Nachdem das Projekt erstellt ist, wird <code>tlbimp.exe</code> ausgeführt, um eine Interop-Assemblierung zu generieren.

Elemente der Gruppe "Linker-Ausgabe"	Beschreibung
DCUs erzeugen	Erstellt <code>.dcu</code> -Dateien im Delphi-Standardformat.
C-Objektdateien erzeugen	Erzeugt eine C-Objektdatei für das Linken mit einem C-Programm (ohne Namensverkürzung).
C++ Objektdateien erzeugen	Erzeugt eine C++ Objektdatei für das Linken in C++ (mit C++ Namensverkürzung).
Mit Namespaces	Bezieht C++-Namespace-Informationen in die erzeugten OBJ- und HPP-Dateien ein.
Alle Symbole exportieren	Bezieht Symbolinformationen in die erzeugten OBJ- und HPP-Dateien ein.
Header-Dateien erzeugen	Bezieht Header-Dateiinformationen in die erzeugten OBJ- und HPP-Dateien ein.

Alle C++-Builder-Dateien erzeugen	Wählen Sie diese Option, um alle Namespace-, Symbol- und Header-Dateiinformationen in das Package einzubeziehen. Diese Option ist nur für Packages von Bedeutung. Es wird empfohlen, diese Option anstelle der Optionen unter C++ Objektdateien erzeugen zu verwenden.
-----------------------------------	--

2

Elemente der Gruppe "EXE- und DLL-Optionen"	Beschreibung
Konsolenanwendung	Veranlasst den Linker in der .exe Anwendungsdatei ein Flag zu setzen, um eine Konsolenanwendung anzuzeigen.
Mit TD32/.PDB-Debug-Info	Veranlasst den Compiler zur Erzeugung von Debug-Informationen, die bei der nächsten Ausführung der Anwendung in eine Programmdatenbankdatei eingefügt werden.
.DRCIL-Datei erzeugen	Erzeugt eine .drcil-Datei, die String-Ressourcen enthält, die in eine Ressourcendatei kompiliert werden können.

Elemente der Gruppe "Speichergrößen"	Beschreibung
Min. Stack-Größe	Gibt die minimale Größe des Stack an (nur bei ausführbaren Projektdateien verwendbar -- für DLLs deaktiviert) Einstellungen zur Speichergröße können auch im Quelltext mit der \$M Compiler-Direktive vorgenommen werden.
Max. Stack-Größe	Gibt die maximale Größe des Stack an (nur bei ausführbaren Projektdateien verwendbar -- für DLLs deaktiviert) Einstellungen zur Speichergröße können auch im Quelltext mit der \$M Compiler-Direktive vorgenommen werden.
Image-Basis	Gibt die Adresse an, an der die ausführbare Datei in den Speicher geladen werden soll. Dieser Wert wird üblicherweise nur beim Compilieren von DLLs geändert.

Elemente der Gruppe "Beschreibung"	Beschreibung
EXE-Beschreibung	Geben Sie in dieses Feld eine Beschreibung des Programms ein (bis zu 255 Zeichen). Dieser String wird mit dem Schalter \$D gelinkt und in die ausführbare Datei eingebunden. Er wird meistens für Copyright-Informationen verwendet. Copyright-Informationen können auch in die Versionsinfo-Datei eingefügt werden. Diese Option ist nur für gemeinsam genutzte Objekte und für ausführbare Anwendungsdateien verwendbar, nicht für Packages.

Allgemeine Elemente	Beschreibung
Standard für neue Projekte	Speichert die aktuellen Einstellungen als Standardeinstellungen für neue Projekte.

2.212 Projektoptionen

Projekt>Optionen

Verwenden Sie dieses Dialogfeld zum Festlegen der Projektoptionen.

Klicken Sie im linken Bereich auf einen Eintrag in einer Kategorie, um dessen Optionsseite anzuzeigen. Wenn Sie den Cursor auf einen Optionsnamen setzen, wird ein Kurzhinweis angezeigt, der eine Optionsbeschreibung, den Standardwert der Option und die Befehlszeilenoption (falls vorhanden) enthält.

Jede Konfiguration, außer **Base**, basiert auf einer anderen Konfiguration. Der Knoten Build-Konfigurationen unter dem Projekt im Fenster Projektverwaltung repräsentiert die Konfiguration **Base**. Alle Konfigurationen sind unter dem Knoten Build-Konfigurationen in einer hierarchischen Liste aufgeführt, die die Über-/Untergeordnet-Beziehung für jede Konfiguration zeigt.

Wenn der Wert einer Option von der übergeordneten Konfiguration abweicht, wird der zugehörige Text in Fettschrift dargestellt. Um den übergeordneten Konfigurationswert wiederherzustellen, klicken Sie den Optionstext mit der rechten Maustaste an und wählen im Kontextmenü Zurücksetzen. Wenn Sie Optionswerte ändern, können Sie Ihre Änderungen in einer neuen Konfiguration oder einer benannten Optionsgruppe speichern. Sie können zu dieser Konfiguration wechseln, oder diese Optionsgruppe in die aktive Konfiguration laden.

Build-Konfigurationsoptionen	Beschreibung
Build-Konfiguration	<p>Zeigt die aktive Build-Konfiguration an. Wählen Sie mit Hilfe der Drop-Down-Liste eine andere Build-Konfiguration aus.</p> <p>Die Drop-Down-Liste Build-Konfiguration enthält die Konfigurationen für das jeweilige Projekt — keine Optionsgruppen. Jedes Projekt verfügt — unabhängig von anderen Projekten — über eine eigene Konfigurationsliste.</p> <p>Die Optionswerte einer aktiven Konfiguration können mit der Schaltfläche Speichern unter... als Optionsgruppdatei gespeichert werden. Eine Optionsgruppe kann gesamt oder teilweise mit der Schaltfläche Laden... für die aktive Konfiguration eines Projekts übernommen werden.</p> <p>Es gibt drei Ausgangskonfigurationen: Base, Debug und Ausgabe. Die Konfiguration Base stellt grundlegende Konfigurationsoptionen bereit. Sie kann nicht gelöscht werden, so dass Sie immer über mindestens eine Konfiguration verfügen.</p> <p>Neben einigen Optionen, die eine Liste mit Einträgen enthalten, wie z.B. Definitionen oder Pfadangaben, wird das Kontrollfeld Zusammenführen angezeigt. Wenn dieses Kontrollfeld aktiviert ist, führt die IDE die Optionsliste mit der des unmittelbaren Vorfahren der Konfigurationsliste für diese Option zusammen. Beachten Sie, dass die IDE den Inhalt der Option nicht eigentlich ändert, sondern sich so verhält, als ob die Liste die Liste des Vorfahren enthielte. Wenn das Kontrollfeld Zusammenführen des Vorfahren auch aktiviert ist, führt die IDE auch die Liste dieses Vorfahren für diese Option zusammen, und weiter die Vererbungskette hinauf. Ist das Kontrollfeld deaktiviert, verwendet die IDE nur die Einträge aus der aktuellen Konfiguration.</p>
Speichern unter...	<p>Zeigt der Dialogfeld Speichern unter an, um die Optionen der aktuellen Konfiguration in eine Datei zu speichern, die als benannte Optionsgruppe geladen werden kann.</p> <p>Nur die Werte in der aktuellen, aktiven Konfiguration, die sich von der übergeordneten Konfiguration unterscheiden, werden in der Optionsgruppdatei gespeichert. Dazu gehören alle Werte, die Sie in der aktiven Konfiguration geändert haben, als auch alle Werte, deren Standardwerte bereits in der Konfiguration verändert waren.</p>

Laden...	Zeigt das Dialogfeld Optionsgruppe anwenden an, in dem Sie die Optionen einer benannten Optionsgruppe für die aktuelle Konfiguration übernehmen können. Sie haben drei Auswahlmöglichkeiten für das Laden der Werte: Überschreiben, Ersetzen oder Beibehalten
----------	--

Allgemeine Option	Beschreibung
Standard für neue Projekte	Speichert die aktuellen Einstellungen als Standardeinstellungen für neue Projekte.

Siehe auch

Optionsgruppe anwenden (Dialogfeld) ( siehe Seite 492)

Überblick zu Build-Konfigurationen ( siehe Seite 1463)

Manager für Build-Konfigurationen ( siehe Seite 496)

Arbeiten mit benannten Optionsgruppen ( siehe Seite 15)

Übernehmen der aktiven Build-Konfiguration ( siehe Seite 2)

2.213 Packages

Projekt ▶ Optionen ▶ Packages

Verwenden Sie diese Seite, um die Entwurfszeit-Packages, die in der IDE installiert sind, sowie die Laufzeit-Packages, die vom Projekt benötigt werden, festzulegen.

Element	Beschreibung
Entwurfs-Packages	Führt die Entwurfszeit-Packages auf, die in der IDE und für alle Projekte verfügbar sind. Markieren Sie ein Entwurfszeit-Package, das für das aktuelle Projekt verfügbar gemacht werden soll.
Hinzufügen	Installiert ein Entwurfszeit-Package. Das Package wird dem aktuellen Projekt hinzugefügt.
Entfernen	Löscht das ausgewählte Package. Das Package steht für das aktuelle Projekt nicht zur Verfügung.
Bearbeiten	Öffnet das ausgewählte Package im Package-Editor, wenn der Quelltext oder die dcp-Datei verfügbar sind.
Komponenten	Zeigt eine Liste der Komponenten an, die im ausgewählten Package enthalten sind.
Laufzeit-Packages	Gibt die Laufzeit-Packages an, die beim Erstellen der ausführbaren Datei verwendet werden. Trennen Sie Package-Namen durch Semikolons. Immer wenn Packages installiert und deinstalliert werden, wird die Liste der Runtime-Packages entsprechend aktualisiert. Laufzeit-Packages, die für installierte Entwurfszeit-Packages erforderlich sind, werden automatisch hinzugefügt.
Laufzeit-Packages verwenden	Linkt die Laufzeit-Packages in Ihrem Projekt dynamisch und aktiviert das Laufzeit-Packages-Eingabefeld.
Ellipsen-Schaltflächen	Zeigt das Dialogfeld Laufzeit-Package an, in dem Sie den Namen eines Laufzeit-Package angeben können, das in die Liste Laufzeit-Packages aufgenommen werden soll.
Standard für neue Projekte	Speichert die aktuellen Einstellungen als Standardeinstellungen für neue Projekte.

2.214 Pre-Build- oder Post-Build-Ereignis

Verwenden Sie dieses Dialogfeld zum Erstellen einer Liste mit Befehlen und Makros, die an bestimmten Punkten im Build-Prozess ausgeführt werden sollen. Geben Sie in der Liste beliebige, gültige DOS-Befehle an. Die Befehle und deren Ergebnisse werden auf der Registerkarte Ausgabe des Fensters Meldungen angezeigt.

Projekt▸**Optionen**▸**Build-Ereignisse**▸**Bearbeiten...**

Element	Beschreibung
Ausführen wenn	Führt die angegebenen Ereignisse immer aus oder nur wenn das Ziel veraltet ist. Diese Option betrifft nur Post-Build-Ereignisse.
Build bei Fehler abbrechen	Bricht das Build-Projekt ab, wenn ein Befehl einen Fehlercode zurückgibt, der nicht Null ist.
Anweisungen	Geben Sie die Anweisung an, die ausgeführt werden soll. Trennen Sie jede Anweisung durch ein Zeichen für eine neue Zeile. Klicken Sie auf die Schaltfläche Bearbeiten, um ein Dialogfeld anzuzeigen, das Sie beim Erstellen einer Ereignisliste unterstützt.
Makros	Listet für Sie verfügbare Makros auf, die als Befehlsargumente verwendet werden können.

2.215 Signatur

Projekt ▶ Optionen ▶ Signatur

Verwenden Sie diese Seite, um die vom Projekt erzeugte Assemblierung mit dem Microsoft-Hilfsprogramm Strong Name (SN) mit einer Signatur zu versehen.

Anmerkung: Signaturen können nur im .NET-Framework verwendet werden.

Element	Beschreibung
Assemblierung signieren	Versehen Sie die vom Projekt erzeugte Assemblierung (ausführbare Datei, DLL, etc.) mithilfe des Microsoft-Hilfsprogramms Strong Name mit einer Signatur. Weitere Informationen über Signaturen finden Sie im Microsoft .NET SDK.
Schlüsseldatei für starke Namen	Legt die Datei mit der Signatur fest. Klicken Sie auf die Ellipsen-Schaltfläche, um ein Dialogfeld für die Suche nach der Datei zu öffnen.
Nur verzögerte Signatur	Verzögert das Signieren der vom Projekt erzeugten Assemblierung. Verwenden Sie diese Option nur, wenn nach dem Build noch Bearbeitungen an der Assemblierung vorgenommen werden müssen. Diese Option ist standardmäßig deaktiviert, weil ohne Signatur die Assemblierung nicht ausgeführt oder gedebuggt werden kann.

Warnung: Wenn Sie die Assemblierung mit einer Signatur versehen, verhindert das Verzögern des Signierens, dass die Assemblierung ausgeführt oder gedebuggt werden kann.

2.216 Debugger-Symboltabellen

Projekt▸**Optionen**▸**Debugger**▸**Symboltabellen**. Verwenden Sie dieses Dialogfeld, um den Speicherort der Symboltabellen anzugeben, die bei der Fehlersuche verwendet werden sollen. Dieses Dialogfeld lässt sich auch über **Start**▸**Parameter** aufrufen.

Element	Beschreibung
Suchpfad für Debug-Symbole	Gibt das Verzeichnis an, das die Symboltabellen enthält, die bei der Fehlersuche verwendet werden sollen. Wenn das Kontrollfeld Alle Symbole laden aktiv ist, wird dieser Pfad verwendet.
Alle Symbole laden	Definiert den Status der Liste für die Zuordnungen des Modulnamens zum Symboltabellenpfad. Ist das Kontrollfeld aktiviert, ist die Auswahlliste deaktiviert und alle Symboltabellen werden vom Debugger geladen. Der Debugger verwendet den Suchpfad für Debug-Symbole, um nach der jeweils zu dem Modul gehörenden Symboltabellendatei zu suchen, das von dem aktuell zu überprüfenden Prozess geladen wird. Ist das Kontrollfeld nicht aktiviert, ist die Liste für die Zuordnungen des Modulnamens zum Symboltabellenpfad aktiv und die dortigen Einstellungen werden benutzt.
Zuordnungen des Modulnamens zum Symboltabellenpfad	Zeigt die aktuelle Zuordnung jedes Modulnamens zum Suchpfad einer Symboltabelle an, die für das Projekt definiert ist. Mit den nach oben bzw. nach unten weisenden Pfeilen rechts neben der Liste mit den Suchpfaden können Sie einen markierten Pfad in der Suchreihenfolge nach oben bzw. nach unten verlagern. Der Debugger durchsucht diese Liste, um eine Übereinstimmung für den Namen des zu ladenden Moduls zu finden. Hat der Debugger einen entsprechenden Modulnamen gefunden, verwendet er den zugehörigen Pfad, um die Symboltabelle dieses Moduls zu suchen. Angenommen, das Modul foo123.dll ist geladen und die Liste zeigt foo*.dll als erstes Element und *123.dll als ein nachfolgendes Element an, so verwendet der Debugger nur den Symboltabellenpfad für foo*.dll , auch wenn beide Elemente dem Modul entsprechen, das gerade geladen wird.
Symbole für unspezifizierte Module laden	Gibt an, ob Symboltabellen für Module, die nicht in der Liste der Zuordnungen des Modulnamens zum Symboltabellenpfad enthalten sind (entweder explizit oder über eine Dateimaske), bei der Fehlersuche geladen werden. Ist das Kontrollfeld aktiv, werden die Symboltabellen für nicht definierte Module über den Suchpfad für Debug-Symbole geladen. Ist das Kontrollfeld nicht aktiv, werden die Symboltabellen nur für die Module in der Liste geladen.
Neu	Zeigt das Dialogfeld Suchpfad für Symboltabelle hinzufügen an, in dem Sie einen Modulnamen und einen zugehörigen Suchpfad für Tabellen angeben können. Das Modul und der Pfad werden der Liste der Zuordnungen des Modulnamens zum Symboltabellenpfad hinzugefügt. Beachten Sie, dass Sie einen leeren Pfad hinzufügen können, um zu verhindern, dass eine Symboltabelle für ein Modul geladen wird.
Bearbeiten	Zeigt das ausgewählte Modul und den Pfad im Dialogfeld Suchpfad für Symboltabelle hinzufügen an, wodurch Sie den Modulnamen oder Pfad bearbeiten können, der in der Liste der Modulzuordnungen angezeigt wird.
Löschen	Entfernt das ausgewählte Modul aus der Liste der Zuordnungen des Modulnamens zum Symboltabellenpfad.
Standard für neue Projekte	Speichert die aktuellen Einstellungen als Standardeinstellungen für neue Projekte.

Siehe auch

Anwendungen debuggen (siehe Seite 1439)

Dateien für das externe Debuggen vorbereiten (siehe Seite 35)

2.217 Versionsinformationen

Projekt>Optionen>Versionsinformationen

Verwenden Sie dieses Dialogfeld, um Versionsinformationen für ein Delphi-Win32-Projekt festzulegen. Wenn der Benutzer dann mit der rechten Maustaste auf das Programmsymbol klickt und den Befehl Eigenschaften wählt, werden die Versionsinformationen angezeigt.

Element	Beschreibung
Versionsinformationen in das Projekt übernehmen	Legt fest, ob der Anwender Produktidentifikationsinformationen einsehen kann. Damit diese Option in Konsolanwendungen verfügbar ist, müssen Sie {\$R *.res} in Ihre Projektquelle aufnehmen.
Versionsnummer des Moduls	Legt eine hierarchisch strukturierte Angabe zu Haupt- und Nebenversion, Ausgabe und Compilierung fest. In den Feldern Hauptversion, Nebenversion, Ausgabe und Build können vorzeichenlose Ganzzahlen im Bereich von 0 bis 65.535 eingegeben werden. Der daraus zusammengesetzte String bildet eine Versionsnummer für die Anwendung, beispielsweise 2.1.3.5. Wenn Sie die Option Build-Nummer automatisch erhöhen aktivieren, wird die Build-Nummer jedes Mal erhöht, wenn der Menübefehl Projekt>Build <Projekt> gewählt wird. Bei anderen Formen der Compilierung wird diese Nummer nicht erhöht.
Modulattribute	Gibt den vorgesehenen Einsatzzweck dieser Version an: Fehlersuche, Testversion usw. Modulattribute können zu Informationszwecken in die Versionsinformationen aufgenommen werden. Wenn ein Projekt im Debug-Modus compiliert wird, wird das entsprechende Flag in die Versionsinformationen aufgenommen. Die verbleibenden Flags können nach Bedarf gesetzt werden. Debug-Build Das Projekt wurde im Debug-Modus compiliert. Testversion Die Version ist keine endgültige Version. DLL Das Projekt enthält eine DLL. Spezielles Build Die Version ist eine Abwandlung der Standardversion. Privates Build Die Version wurde nicht unter den üblichen Gesichtspunkten für Freigabeverisionen compiliert.
Sprache	Gibt an, welche Codepage zur Ausführung der Anwendung auf dem System des Benutzers erforderlich ist. Sie können hier nur eine Sprache auswählen, die in der Registerkarte Ländereinstellung der Systemsteuerung Ihres Computers aufgeführt ist. In einigen Windows-Versionen werden nicht alle Landessprachen (z. B. fernöstliche Sprachen) unterstützt. Sie müssen mit Hilfe des entsprechenden Sprachpaketes separat installiert werden.
Liste Schlüssel/Wert	Legt typische Daten zur Produktidentifikation fest. Die Schlüsseleinträge können geändert werden, indem der Schlüssel markiert und der Name neu eingegeben wird. Zum Hinzufügen von Schlüsseleinträgen klicken Sie mit der rechten Maustaste in die Tabelle und wählen den Befehl Schlüssel hinzufügen.
Standard für neue Projekte	Speichert die aktuellen Einstellungen als Standardeinstellungen für neue Projekte.

Tip: Um programmseitig Versionsinformationen Ihrer compilierten Win32-Anwendung zu ermitteln, verwenden Sie die Windows-API-Funktionen `GetFileVersionInfo` und `VerQueryValue`.

2.218 Ordner- oder Verzeichnisansicht hinzufügen

In diesem Dialogfeld können Sie dem aktiven Projekt einen Ordnerknoten hinzufügen. Sie können einen Ordnerknoten oder eine Verzeichnisansicht dazu verwenden, um durch häufig benötigte Dateien zu blättern, die nicht Teil Ihres Projekts sind.

Element	Beschreibung
Speicherort	Gibt den Pfad des neuen Ordnerknotens an. Der Speicherort kann ein absoluter oder relativer Pfad sein. Sie können auf [...] klicken, um zu einem Ordner zu navigieren.
Dateitypen	Mit diesem Filter können Sie nicht benötigte Dateien ausblenden. Standardmäßig ist für den Filter Dateitypen die Einstellung * . * aktiv, die alle Dateien anzeigt.
Unterverzeichnisse anzeigen	Zeigt Unterverzeichnisse im Ordnerknoten an. Wenn diese Option deaktiviert ist, werden nur die Dateien angezeigt.

2.219 ATL

Projekt ▶ Optionen ▶ ATL

Verwenden Sie dieses Dialogfeld, um die ATL-Optionen anzugeben. Diese Optionen gelten für jeden COM-Server im Projekt.

Element	Beschreibung
Einfach	Für jeden Client wird eine Instanz des COM-Serverobjekts erstellt.
Mehrfach	Alle Clients arbeiten mit derselben Instanz des COM-Servers.
APARTMENTTHREADED	Ein Objekt wird nur durch den Thread referenziert, in dem es instantiiert wurde. Diese Option sollte für Projekte eingesetzt werden, die nur Einzel-Thread- und Apartment-Thread-Objekte enthalten.
MULTITHREADED	Auf Objekte lässt sich von jedem Thread aus referenzieren. Verwenden Sie diese Option für Projekte, die Objekte des freien Thread-Modells (beidseitig oder neutral) enthalten.
Single	Alle COM-Server werden mit einem einzigen Thread implementiert.
Apartment	Auch wenn das Projekt multithread-fähig ist, benötigt jede Instanz des COM-Servers einen eigenen Thread für OLE-Aufrufe.
Frei	Das Projekt kann multithread-fähig sein, und jede Instanz des COM-Server-Objekts kann gleichzeitige Aufrufe von verschiedenen Threads erhalten. (Sie müssen dazu für Thread-Gleichzeitigkeit sorgen.)
Beides	Entspricht Frei, außer dass ausgehende Aufrufe, wie z.B. Callbacks, garantiert im selben Thread ausgeführt werden.
Neutral	Mehrere Clients können das Objekt gleichzeitig in verschiedenen Threads aufrufen, aber COM stellt sicher, dass die Aufrufe nicht miteinander in Konflikt geraten.
QueryInterface verfolgen	Schickt eine Nachricht an das Ereignisprotokoll, sobald der Client einen QueryInterface-Aufruf ausführt. Dort wird auch der Status des Aufrufs angezeigt.
Ref-Zähler überprüfen	Bei jeder Erhöhung oder Erniedrigung des Referenzzählers eines COM-Servers wird eine Nachricht an das Ereignisprotokoll gesendet. Wenn die Option Ref-Zähler überprüfen aktiviert ist und das Projekt versucht, das Objekt mit einem Referenzzähler ungleich 0 aus dem Speicher zu entfernen, tritt eine Assertion auf.
Allgemeine Ablaufverfolgung	Wenn eine ATL-Funktion aufgerufen wird, wird eine Nachricht an das Ereignisprotokoll geschickt.

Anmerkung: Threading-Modell-Optionen sind aus Gründen der Abwärtskompatibilität beibehalten worden. Sie können die Threading-Modell auf Objektbasis angeben.

Siehe auch

[ATL-Überblick \(MSDN\)](#)

2.220 C++-Compiler Erweiterte Compilierung

Projekt ▶ Optionen ▶ C++-Compiler ▶ Erweiterte Compilierung

Verwenden Sie dieses Dialogfeld dazu, erweiterte Compilieroptionen des C++-Compilers zu setzen.

Build-Konfigurationsoptionen	Beschreibung
Build-Konfiguration	Zeigt die aktive Build-Konfiguration an. Sie können das Drop-Down-Menü dazu verwenden, eine andere Build-Konfiguration auszuwählen.
Speichern unter...	Zeigt das Dialogfeld Speichern unter an, um die Optionen der aktuellen Konfiguration in eine Datei zu speichern, die als benannte Optionsgruppe geladen werden kann.
Laden...	Zeigt das Dialogfeld Optionsgruppe an, in dem Sie die Optionen einer benannten Optionsgruppe für die aktuelle Konfiguration übernehmen können.

Gleitkommaoptionen	Beschreibung
Schnelles Gleitkomma (-ff)	<p>Gleitkommaoperationen können ohne Berücksichtigung expliziter oder impliziter Typkonvertierungen optimiert werden. Berechnungen verlaufen dann schneller als im ANSI-Modus.</p> <p>Diese Option dient dazu, bestimmte Optimierungen zu ermöglichen, die auf technischer Ebene keiner korrekten C-Semantik entsprechen.</p> <pre>double x; x = (float) (3.5*x);</pre> <p>Um dies korrekt ausführen zu können, wird x mit 3,5 multipliziert. Das daraus resultierende Double wird entsprechend der Gleitkommagenaugkeit gekürzt und dann als Double in x gespeichert. Bei der schnellen Gleitkommaoperation wird das lange Double-Produkt direkt in ein Double konvertiert. Da nur wenige Programme auf diese hohe Genauigkeit angewiesen sind, ist das schnelle Gleitkomma die Standardeinstellung.</p> <p>Wenn diese Option nicht markiert ist (-ff), folgt der Compiler strikt den ANSI-Regeln zur Gleitkommakonvertierung.</p> <p>(Vorgabe = aktiviert)</p>
FDIV-Fehler korrigieren (-fp)	<p>Einige Pentium-Prozessoren der ersten Serie führen bestimmte Gleitkommadivisionen nicht mit voller Genauigkeit aus. Auch wenn die Chancen gering sind, dass dieses Problem heute noch auftaucht, kann mit dieser Option Code eingefügt werden, der eine Gleitkommaoperation so emuliert, dass sicher das richtige Ergebnis geliefert wird. Die Geschwindigkeit bei Ausführung des Prozessorbefehls FDIV sinkt dann allerdings.</p> <p>Diese Option korrigiert FDIV-Befehle nur in Modulen, die Sie compilieren. Die Laufzeitbibliothek enthält ebenfalls FDIV-Befehle, die von dieser Option nicht modifiziert werden. Wenn Sie das Problem auch in den Laufzeitbibliotheken beheben möchten, müssen Sie sie mit dieser Einstellung neu compilieren.</p> <p>Die folgenden Funktionen verwenden FDIV-Befehle in Assemblersprache und werden von dieser Option nicht erfasst: acos, acosl, acos, asinasinl, atanatan2, atan2latanl, coscosh, cosh1cosl, expexpl, fmodfmodl, powpow10, pow10lpowl, sinsinh, sinhlsinl, tantanh, tanhtanh</p> <p>Ähnliches gilt für Funktionen, die eine Gleitkommazahl in einen String konvertieren oder umgekehrt (z.B. printf oder scanf).</p> <p>(Vorgabe = nicht aktiviert)</p>
Stille Gleitkommavergleiche (-fq)	Verwendet die stille Gleitkommaanweisung (FUCOMP). (Vorgabe = aktiviert)

String-Optionen	Beschreibung
Schreibbare Strings (-dw)	Stellt für Strings zugewiesenen Speicher in das beschreibbare Datensegment. (Vorgabe = nicht aktiviert)
Schreibgeschützte Strings (-dc)	Stellt für Strings zugewiesenen Speicher in das schreibgeschützte Datensegment. (Vorgabe = nicht aktiviert)
Doppelte Strings zusammenführen (-d)	Verbindet zwei literale Strings, wenn einer mit dem anderen identisch ist. Daraus entstehen kleinere Programme, die jedoch etwas mehr Zeit zum Compilieren benötigen. Fehler können dann entstehen, wenn einer der Strings verändert wird. (Vorgabe = nicht aktiviert)

Weitere Optionen	Beschreibung
Codeseite (-CP)	<p>Bietet Unterstützung für benutzerdefinierte Codeseiten. Damit wird dem Compiler mitgeteilt, wie Multibyte-Zeichensätze (MBCS) zerlegt und konvertiert werden sollen.</p> <p>Es gibt zwei bestimmte Bereiche, wo Codeseiten eine Rolle spielen:</p> <ol style="list-style-type: none"> 1. String-Konstanten, Kommentare, #error- und #pragma-Direktiven 2. "Wide-character" String-Konstanten (wie durch L'<MBCS string>' angegeben) <p>Für MBCS-Strings, die zur ersten Gruppe gehören, müssen Sie die korrekte Codeseite mit einem Aufruf der Windows API-Funktion IsDBCSLeadByteEx angeben. Mit dieser Funktion geben Sie die Codeseite an, um die MBCS-Strings für eine bestimmte Gebietseinstellung korrekt zu analysieren (z.B. wird der Compiler befähigt, Backslashes korrekt in nachgestellte MBCS-Bytes zu zerlegen).</p> <p>Für MBCS-Strings, die zur zweiten Gruppe gehören ("Wide-character" String-Konstanten), geben Sie die korrekte Codeseite an, um die MBCS-Strings mit der Windows-API-Funktion MultiByteToWideChar in Unicode-Strings umzuwandeln.</p> <p>Syntax</p> <p>Aktivieren Sie das Code-Paging mit dem folgenden Befehlszeilenschalter:</p> <p>-CPnnnn</p> <p>In dieser Syntax ist <i>nnnn</i> der Dezimalwert der Codeseite, die für eine bestimmte Gebietseinstellung verwendet werden soll.</p> <p>Es gelten die folgenden Regeln:</p> <ol style="list-style-type: none"> 1. Bei der Einstellung des Code-Paging müssen die numerischen Einstellungen für <i>nnnn</i> zu den Werten der Microsoft NLS Code Page ID passen. Zum Beispiel: Verwenden Sie 437 für MS-DOS-Anwendungen aus den Vereinigten Staaten. Verwenden Sie 932 für japanische. 2. Der numerische Wert <i>nnnn</i> muss eine gültige Codeseite sein, die vom BS unterstützt wird. 3. Benutzer müssen eventuell die relevanten Windows NLS-Dateien installieren, um sicherzustellen, dass asiatische Gebietseinstellungen und Codeseiten zugänglich sind. Nähere Informationen hierzu finden Sie in der Dokumentation zu Microsoft NLS Code Page. 4. Wenn Sie keinen Codeseitenwert angegeben haben, ruft der Compiler die Windows-API-Funktion GetACP auf, um die Standard-Codeseite des Systems zu laden. Dieser Wert wird dann verwendet, wenn Strings wie oben dargestellt behandelt werden müssen. <p>Die Vorgabe ist, keine Codeseite zu verwenden.</p>
Weitere Optionen	Alle weiteren Optionen für die Übergabe an den Compiler.
Vorzeichenloser Standardzeichensatztyp (-K)	Der Compiler behandelt die char -Deklarationen so, als würde es sich um einen unsigned char -Typ handeln, der für Kompatibilität mit anderen Compilern sorgt. (Vorgabe = nicht aktiviert)

Quelltextoptionen	Beschreibung
Bezeichnerlänge (-i)	<p>Gibt die Anzahl signifikanter Zeichen (jene, die vom Compiler erkannt werden) in einem Bezeichner an.</p> <p>Mit Ausnahme von C++, wo es für Bezeichner keine Längenbeschränkung gibt, werden Bezeichner nur dann als unterschiedlich angesehen, wenn sie sich in den signifikanten Zeichen unterscheiden. Dies betrifft Variablen, Namen von Präprozessor-Makros und Namen von Strukturelementen.</p> <p>Gültige Werte für die Länge sind 0 und 8 bis 250, wobei 0 die Maximallänge von 250 Zeichen bedeutet.</p> <p>C++Builder unterscheidet standardmäßig 250 Zeichen pro Bezeichner. Andere Systeme (darunter auch einige UNIX-Compiler) ignorieren das neunte und alle folgenden Zeichen. Wenn Sie eine Anwendung auf andere Systeme portieren wollen, sollten Sie den Quelltext mit einer kleineren Anzahl signifikanter Zeichen compilieren. So können Sie Namenskonflikte in langen Bezeichnern aufspüren, die abgeschnitten wurden.</p>
Verschachtelte Kommentare aktivieren (-C)	<p>Schachtelt Kommentare in Ihren C- und C++-Quelltextdateien.</p> <p>Solche Kommentare sind in Standard-C-Implementierungen nicht erlaubt, und sie sind auch nicht portierbar.</p> <p>(Vorgabe = nicht aktiviert)</p>

Allgemeine Option	Beschreibung
Standard für neue Projekte	Speichert die aktuellen Einstellungen als Standardeinstellungen für neue Projekte.

2.221 C++-Compiler C++-Kompatibilität

Projekt>Optionen>C++-Compiler>C++-Kompatibilität

Verwenden Sie dieses Dialogfeld dazu, um spezielle Kompatibilitätsoptionen für den C++-Compiler festzulegen.

Diese Optionen ermöglichen die Rückwärtskompatibilität mit früheren Versionen des Compilers. Im Allgemeinen sollten Sie diese Optionen nur aktivieren, wenn Sie diese Kompatibilität auch wirklich benötigen. Der Vorgabewert für diese Optionen ist false.

Build-Konfigurationsoptionen	Beschreibung
Build-Konfiguration	Zeigt die aktive Build-Konfiguration an. Wählen Sie mit Hilfe der Drop-Down-Liste eine andere Build-Konfiguration aus.
Speichern unter...	Zeigt das Dialogfeld Speichern unter an, um die Optionen der aktuellen Konfiguration in eine Datei zu speichern, die als benannte Optionsgruppe geladen werden kann.
Laden...	Zeigt das Dialogfeld Optionsgruppe an, in dem Sie die Optionen einer benannten Optionsgruppe für die aktuelle Konfiguration übernehmen können.

Anmerkung: Es gibt einige Kompatibilitätsoptionen, deren Schalter mit **-Vb** beginnen: **-Vbe**, **-Vbn**, **-Vbo**, **-Vbr**, **-Vbs** und **-Vbx**. Sie können alle mit dem Schalter **-Vb** gesetzt und zurückgesetzt werden: **-Vb** und **-Vb+** aktiviert alle **-Vb**-Schalter, **-Vb-** deaktiviert alle **-Vb**-Schalter und **-Vb.** setzt alle **-Vb**-Schalter auf ihre Standardwerte.

C++-Kompatibilitätsoptionen	Beschreibung
Nicht konstante Aufrufe für konstantes Objekt (-Vbn)	Lässt den Aufruf einer nicht konstanten Member-Funktion für ein konstantes Objekt zu. Vorgabe = nicht aktiviert
Alte Auflösung für das Überladen (-Vbo)	Verwendet die alten Auflösungsregeln für das Überladen. Vorgabe = nicht aktiviert
Nicht konstante Referenzbindung (-Vbr)	Lässt nicht konstante Referenzbindung zu. Vorgabe = nicht aktiviert
Explizite Template-Spezialisierung als Member-Funktion (-Vbx)	Verwendet die explizite Template-Spezialisierung als Member-Funktion. Vorgabe = nicht aktiviert
Explizite Template-Spezialisierung im alten Stil (-Vbe)	Lässt die explizite Template-Spezialisierung im alten Stil zu. Vorgabe = nicht aktiviert
Klassenargumente im alten Stil (-Va)	Unterstützt Klassenargumente im alten Stil. Vorgabe = nicht aktiviert
Konstruktor-Verschiebungen (-Vc)	Unterstützt Konstruktor-Verschiebungen. Vorgabe = nicht aktiviert

2	Alter Gültigkeitsbereich für for-Anweisung (-Vd)	<p>Gibt den Gültigkeitsbereich der Variablen für Schleifen-Ausdrücke an. Die Ausgabe der folgenden Codesegmente ändert sich je nach Einstellung dieser Option.</p> <pre>int main(void) { for(int i=0; i<10; i++) { cout << "Inside for loop, i = " << i << endl; } //Ende des for-Schleifenblocks cout << "Outside for loop, i = " << i << endl; //Fehler ohne -Vd } //Ende des Blocks, der die for-Schleife enthält</pre> <p>Ist diese Option nicht aktiviert (Standardeinstellung), erstreckt sich die Variable <i>i</i> über den Gültigkeitsbereich hinaus, wenn die Verarbeitung das Ende der Schleife erreicht. Deshalb erhalten Sie den Compilierungsfehler Undefiniertes Symbol, wenn Sie diesen Code mit dieser deaktivierten Option compilieren.</p> <p>Ist diese Option aktiviert (-Vd), erstreckt sich die Variable <i>i</i> über den Gültigkeitsbereich hinaus, wenn die Verarbeitung das Ende der Schleife erreicht. In diesem Fall würde die Ausgabe so lauten:</p> <pre>Inside for loop, i = 0 ... Outside for loop, i = 10</pre> <p>Vorgabe = nicht aktiviert</p>
	Altes Borland-Klassen-Layout (-VI)	Dies ist eine Option für die Rückwärtskompatibilität. Sie veranlasst den C++-Compiler dazu, abgeleitete Klassen auf dieselbe Weise zu generieren, wie das in früheren Versionen von C++Builder der Fall war. Aktivieren Sie diese Option, wenn Sie Quelldateien compilieren möchten, die zusammen mit älteren Versionen von C++Builder verwendet werden sollen (z.B. wenn Sie mit einer .DLL arbeiten möchten, die sich nicht neu compilieren lässt, oder wenn Sie ältere Datendateien verwenden, die hartcodierte Klassen-Layouts enthalten). Vorgabe = nicht aktiviert
	'this' zuerst (-Vp)	Forciert, wie bei Pascal, zuerst 'this'. Der Compiler legt Parameter normalerweise von rechts nach links auf den Stack. Vorgabe = nicht aktiviert
	VTable nach vorne (-Vt)	Setzt den virtuellen Tabellenzeiger an den Anfang des Objekt-Layouts.. Vorgabe = nicht aktiviert
	Langsame virtuelle Basiszeiger ('slow') (-Vv)	Verwendet langsame virtuelle Basiszeiger ('slow'). Vorgabe = nicht aktiviert
	Leere Klassenelementfunktionen mit Nulllänge (-Vx)	Meist ist die Größe eines Datenelements in einer Klassendefinition mindestens ein Byte. Wenn diese Option aktiviert ist, ermöglicht der Compiler eine leere Struktur mit Nulllänge. Vorgabe = nicht aktiviert
	Leere Basisklasse mit Nulllänge (-Ve)	Normalerweise ist die Größe einer Klasse mindestens ein Byte, auch wenn die Klasse keine Datenelemente definiert. Wenn Sie diese Option setzen, ignoriert der Compiler dieses nicht verwendete Byte beim Aufbau des Speichers und der Gesamtgröße einer abgeleiteten Klasse; leere Basisklassen verbrauchen keinen Platz in abgeleiteten Klassen. Vorgabe = nicht aktiviert

Allgemeine Option	Beschreibung
Standard für neue Projekte	Speichert die aktuellen Einstellungen als Standardeinstellungen für neue Projekte.

2.222 C++Compiler C++-Compilierung

[Projekt](#) ▶ [Optionen](#) ▶ [C++Compiler](#) ▶ [C++-Compilierung](#)

Verwenden Sie dieses Dialogfeld dazu, Optionen für den C++ Compiler zu setzen.

Build-Konfigurationsoptionen	Beschreibung
Build-Konfiguration	Zeigt die aktive Build-Konfiguration an. Wählen Sie mit Hilfe der Drop-Down-Liste eine andere Build-Konfiguration aus.
Speichern unter...	Zeigt das Dialogfeld Speichern unter an, um die Optionen der aktuellen Konfiguration in eine Datei zu speichern, die als benannte Optionsgruppe geladen werden kann.
Laden...	Zeigt das Dialogfeld Optionsgruppe an, in dem Sie die Optionen einer benannten Optionsgruppe für die aktuelle Konfiguration übernehmen können.

C++-Optionen	Beschreibung
Template-Erzeugung	<p>Standard (-Jgd)*</p> <p>Der Compiler erzeugt public (globale) Definitionen für alle Template-Instanzen. Wenn dieselbe Template-Instanz von mehreren Modulen erzeugt wird, fügt der Linker die Einzelinstanzen automatisch zu einer einzigen Kopie dieser Instanz zusammen.</p> <p>Für die Erzeugung der Instanzen benötigt der Compiler den Funktionsrumpf (bei einer Template-Funktion) oder die Rümpfe der Elementfunktionen und Definitionen für statische Datenelemente (bei einer Template-Klasse). Diese sind in der Regel in einer Header-Datei enthalten.</p> <p>Template-Instanzen lassen sich so einfach erstellen.</p> <p>Dies ist die Standardeinstellung.</p> <p>Extern (-Jgx)</p> <p>Der Compiler erzeugt externe Referenzen zu allen Template-Instanzen.</p> <p>Für Template-Instanzen, die gelinkt werden müssen, muss eine explizite Instantiierungsdirektive in mindestens einem anderen Modul enthalten sein.</p>
Virtuelle Tabellen	<p>Intelligent (-V)*</p> <p>Es werden gemeinsame virtuelle C++-Funktionstabellen und Inline-Funktionen für alle Module in einer Anwendung erzeugt. Das Programm enthält dann jeweils nur eine Instanz für jede virtuelle Tabelle bzw. Inline-Funktion.</p> <p>Mit der Option Intelligent erhalten Sie sehr kleine und effiziente ausführbare Dateien. Die erzeugten .OBJ- und .ASM-Dateien sind aber nur zu Linkern und Assemblern von CodeGear kompatibel.</p> <p>Dies ist die Standardeinstellung.</p> <p>Extern (-V0)</p> <p>Es werden externe Referenzen auf virtuelle Tabellen erzeugt. Wenn Sie die Option Intelligent nicht verwenden wollen, können Sie mit Extern (oder Public) globale virtuelle Tabellen erzeugen und referenzieren.</p> <p>Bei Verwendung von Extern müssen ein oder mehrere Module des Programms mit der Option Public compiliert werden, um die Definitionen für die virtuellen Tabellen bereitzustellen.</p> <p>Public (-V1)</p> <p>Es werden public Definitionen für virtuelle Tabellen erzeugt. Bei Verwendung der Option Extern (-V0) muss mindestens eines der Module im Programm mit der Option Public compiliert werden, um die Definitionen für die virtuellen Tabellen bereitzustellen. Alle anderen Module müssen dann mit Extern compiliert werden, damit sie auf die Public-Kopie der virtuellen Tabellen Bezug nehmen.</p>

Elementzeiger	<p>So klein wie möglich (-Vmd)</p> <p>Member-Zeiger müssen die kleinstmögliche Repräsentation verwenden, damit sie auf alle Member ihrer Klasse zeigen können. Wenn die Klasse an der Stelle der Member-Zeigerdeklaration nicht vollständig definiert ist, verwendet der Compiler die allgemeinste Repräsentation und gibt eine Warnung aus.</p> <p>Dies ist die Standardeinstellung.</p> <p>Mehrfachvererbung (-Vmm)</p> <p>Member-Zeiger können auf Member von Mehrfachvererbungsklassen zeigen. Ausgenommen hiervon sind virtuelle Basisklassen.</p> <p>Einfachvererbung (-Vms)</p> <p>Member-Zeiger können nur auf Member von Basisklassen zeigen, die einfach vererben.</p> <p>Standard *</p> <p>Für die Member-Zeiger werden keine Optionen gesetzt. Dies ist die Standardeinstellung.</p>
Präzision der Elementzeiger (-Vmp)	Der Compiler verwendet die angegebene Genauigkeit für Member-Zeigertypen. Aktivieren Sie diese Option, wenn ein Zeiger auf eine abgeleitete Klasse explizit in einen Member-Zeiger einer einfacheren Basisklasse umgewandelt wird (d.h. der Zeiger zeigt auf ein Element einer abgeleiteten Klasse). Vorgabe = nicht aktiviert

Exception-Behandlungsoptionen	Beschreibung
RTTI aktivieren (-RT)	<p>Der Compiler erzeugt Code, der eine Typidentifikation zur Laufzeit (RTTI) ermöglicht.</p> <p>Allgemein gilt: Wenn die Option Destruktorbereinigung aktivieren (-xd) aktiviert ist, muss auch diese Option gesetzt werden.</p> <p>Vorgabe = aktiviert</p>
Exceptions aktivieren (-x)	<p>Setzt die Exception-Behandlung in C++. Ist diese Option deaktiviert (-x-), und Sie versuchen, die Routinen zu Exception-Behandlung in Ihrem Code zu verwenden, erzeugt der Compiler bei der Compilierung Fehlermeldungen.</p> <p>Wenn Sie diese Option deaktivieren, lassen sich Informationen zur Exception-Behandlung leichter aus Programmen entfernen. Dies ist besonders dann hilfreich, wenn das Programm auf andere Plattformen oder Compiler portiert werden soll.</p> <p>Die Deaktivierung führt nur dazu, dass der Quelltext zur Exception-Behandlung von der Compilierung ausgeschlossen wird. Die Anwendung kann trotzdem Exception-Quelltext enthalten, wenn Sie .OBJ- und Bibliotheksdateien linken, die mit aktivierte Exceptions compiliert wurden (z.B. CodeGear-Laufzeit-Bibliotheken).</p> <p>Vorgabe = aktiviert</p>
Destruktorbereinigung (-xd)	<p>Wenn diese Option aktiviert ist und eine Exception auftritt, werden für alle Objekte, die innerhalb des Bereichs der <code>catch</code>- und <code>throw</code>-Anweisungen deklariert wurden, automatisch die entsprechenden Destruktoren aufgerufen.</p> <p>Normalerweise ist es angebracht, zusammen mit dieser Option auch die Option Laufzeitinformationen aktivieren (-RT) zu setzen.</p> <p>Für dynamische Objekte, die mit <code>new</code> zugewiesen wurden, werden die Destruktoren nicht automatisch aufgerufen (d.h. es erfolgt keine automatische Freigabe dieser Objekte).</p> <p>Vorgabe = aktiviert</p>
Keine DLL/MT-Destruktorbereinigung (-xds)	Führt keine Bereinigung von DLLs oder Multithread-Destruktoren aus. Vorgabe = nicht aktiviert
Schnelle Exception-Prologe (-xf)	Erweitert den Inline-Quelltext für jede Funktion zur Exception-Behandlung. Mit dieser Option können Sie die Performance des Programms erhöhen. Die ausführbare Datei nimmt aber an Umfang zu. Vorgabe = nicht aktiviert

Positionsinformation (-xp)	Ist diese Option aktiv, gibt der Compiler den Dateinamen und die Nummer der Quelltextzeile an, an der eine Exception aufgetreten ist. Dies ermöglicht die Identifizierung von Exceptions zur Laufzeit. Das Programm kann aufgrund dieser Angaben mit __ThrowFileName die Datei und mit __ThrowLineNumber die Zeilennummer ermitteln, in der die Exception aufgetreten ist. Vorgabe = nicht aktiviert
Langsame Exception-Epiloge (-xs)	Wenn diese Option gesetzt ist, wird Epilogcode zur Exception-Behandlung nicht inline erweitert. Diese Option beeinträchtigt die Performance geringfügig. Vorgabe = nicht aktiviert
Exception-Variablen ausblenden (-xv)	<p>Der Compiler behandelt die folgenden Symbole bei der Exception-Behandlung als Sonderfall:</p> <p>__exception_info __exception_code __abnormal_termination</p> <p>Diese werden allen speziellen Compiler/RTL-Konstrukten für eine strukturierte Exception-Behandlung (SEH) zugeordnet. Wenn Sie SEH nicht nutzen und Variablen dieses Namens verwenden, bedeutet dies, dass keine Referenz auf diese Variablen möglich ist, und der Code nicht compiliert wird. -xv bewirkt, dass der Compiler seine speziellen Symbole in diesem Ereignis verbirgt. So lassen sich auch Variablen dieses Namens verwenden.</p> <p>Vorgabe = nicht aktiviert</p>
Globaler Destruktorzähler (-xdg)	Verwendet die globale Destruktur-Zählung (aus Gründen der Kompatibilität mit älteren Versionen des Compilers). Vorgabe = nicht aktiviert

Allgemeine Option	Beschreibung
Standard für neue Projekte	Speichert die aktuellen Einstellungen als Standardeinstellungen für neue Projekte.

2.223 C++ Compiler Debugging

Projekt>Optionen>C++-Compiler>Debugging

Verwenden Sie dieses Dialogfeld dazu, Optionen zur Fehlersuche des C++-Compilers und für CodeGuard festzulegen.

Build-Konfiguration	Zeigt die aktive Build-Konfiguration an. Wählen Sie mit Hilfe der Drop-Down-Liste eine andere Build-Konfiguration aus.
Speichern unter...	Zeigt der Dialogfeld Speichern unter an, um die Optionen der aktuellen Konfiguration in eine Datei zu speichern, die als benannte Optionsgruppe geladen werden kann.
Laden...	Zeigt das Dialogfeld Optionsgruppe an, in dem Sie die Optionen einer benannten Optionsgruppe für die aktuelle Konfiguration übernehmen können.

Optionen zur Fehlersuche	Beschreibung
Debug-Informationen (-v)	<p>Schließt Browser-Informationen in erzeugte .OBJ-Dateien ein. Der Compiler übergibt diese Option an den Linker, damit die Debug-Informationen in die .EXE-Datei eingefügt werden können. Bei der Fehlersuche behandelt diese Option die C++-Inline-Funktionen als normale Funktionen.</p> <p>Sie benötigen die Debug-Informationen, um entweder den integrierten Debugger oder den eigenständigen Turbo Debugger zu verwenden.</p> <p>Ist diese Option nicht aktiv (-v), können Sie größere Objektdateien verknüpfen und erstellen. Diese Option wirkt sich zwar nicht auf die Ausführungsgeschwindigkeit aus, beeinflusst aber die Zeit beim Compilieren und Linken.</p> <p>Ist die Option Zeilennummer aktiv, sollten Sie sicherstellen, dass in den Compiler-Optionen die Einstellung Pentium-Befehlsplan deaktiviert ist. Ist diese Option aktiviert, entspricht der Quelltext nicht exakt den erzeugten Maschinenanweisungen. Das schrittweise Abarbeiten des Quelltextes kann dadurch durcheinander geraten.</p> <p>Vorgabe = nicht aktiviert</p>
Zeilennummern für das Debuggen (-y)	<p>Fügt in die Objekt- und Objektzuordnungsdateien automatisch Zeilennummern ein. Die Zeilennummern werden sowohl vom integrierten Debugger als auch vom Turbo Debugger verwendet.</p> <p>Die Option Debug-Informationen (-v) erzeugt automatisch Zeilennummern-Informationen. Sie können diese Option aber deaktivieren (-v) und stattdessen mit der Option Zeilennummern für das Debuggen (-y) arbeiten, um die Größe der erzeugten Debug-Informationen zu reduzieren. Mit dieser Einstellung können Sie nach wie vor den Quelltext schrittweise abarbeiten, aber Datenelemente lassen sich dann nicht mehr sehen oder prüfen.</p> <p>Das Einbeziehen von Zeilennummern vergrößert die Objekt- und Map-Dateien, beeinträchtigt aber nicht die Ausführungsgeschwindigkeit des Programms.</p> <p>Ist die Option Zeilennummer aktiv, sollten Sie sicherstellen, dass in den Compiler-Optionen die Einstellung Pentium-Befehlsplan deaktiviert ist. Ist diese Option aktiviert, entspricht der Quelltext nicht exakt den erzeugten Maschinenanweisungen. Das schrittweise Abarbeiten des Quelltextes kann dadurch durcheinander geraten.</p> <p>Vorgabe = nicht aktiviert</p>
Inline-Funktionen erweitern (-vi)*	<p>Erweitert C++-Inline-Funktionen.</p> <p>Zum Steuern der Erweiterung von Inline-Funktionen verhält sich die Option Debug-Informationen (-v) bei C++-Code etwas anders: Ist die Erweiterung von Inline-Funktionen deaktiviert, werden die Inline-Funktionen erzeugt und wie jede andere Funktion aufgerufen.</p> <p>Vorgabe = aktiviert</p>

CodeView4 kompatible Debug-Info erzeugen (-v4)	Erzeugt mit CodeView4 kompatible Debug-Informationen. Vorgabe = nicht aktiviert
--	---

CodeGuard(tm)-Optionen	Beschreibung
Alle CodeGuard-Optionen aktivieren (-vG)	Aktiviert alle CodeGuard-Optionen, unabhängig von der CodeGuard-Ebene. Vorgabe = nicht aktiviert
Inline-Zeigerzugriff überwachen (-vGc)	Diese CodeGuard-Option erzeugt Aufrufe zum Überprüfen von Zugriffen im Quelltext. Die Option erkennt fast alle Zeigerfehler. Die Programmausführung wird mit dieser Option fünf- bis zehnmal langsamer. Die Aktivierung dieser CodeGuard-Optionen kann sich wesentlich auf die Ausführungsgeschwindigkeit auswirken. Vorgabe = nicht aktiviert
Globale Stack-Datenzugriffe überwachen (-vGd)	Erstellt die Daten und Stacklayout-Beschreibungen zum schnellen Nachschlagen durch CodeGuard. Anhand dieser Beschreibungen kann CodeGuard Datenverluste und ungültige Zeiger auf lokale, globale und statische Variablen protokollieren. Sie sollten diese Option immer verwenden. Vorgabe = nicht aktiviert
'this'-Zeiger bei Member-Funktionseintritt überwachen (-vGt)	Erzeugt spezielle Epiloge für Member-Funktionen, Konstruktoren und Destruktoren. CodeGuard überprüft den this -Zeiger beim Eintritt in alle Methoden im C++-Quelltext. Diese Option protokolliert Methodenaufrufe zu gelöschten oder ungültigen Objekten, selbst dann, wenn die Methoden selbst nicht auf this zugreifen. Vorgabe = nicht aktiviert

Allgemeine Option	Beschreibung
Standard für neue Projekte	Speichert die aktuellen Einstellungen als Standardeinstellungen für neue Projekte.

Anmerkung: Wenn Sie sowohl das Debugging als auch die Inline-Erweiterung aktivieren möchten, verwenden Sie die Optionen **-v** und **-vi**.

2.224 C++-Compiler C++-Kompatibilität

Projekt ▶ Optionen ▶ C++-Compiler ▶ Allgemeine Kompatibilität

Verwenden Sie dieses Dialogfeld dazu, allgemeine Kompatibilitätoptionen des C++-Compilers zu setzen.

Einige dieser Optionen ermöglichen eine Rückwärtskompatibilität mit früheren Versionen des Compilers. Der Vorgabewert dieser Optionen ist false. Im Allgemeinen sollten Sie diese Optionen nur aktivieren, wenn Sie diese Kompatibilität auch wirklich benötigen.

Build-Konfigurationsoptionen	Beschreibung
Build-Konfiguration	Zeigt die aktive Build-Konfiguration an. Wählen Sie mit Hilfe der Drop-Down-Liste eine andere Build-Konfiguration aus.
Speichern unter...	Zeigt das Dialogfeld Speichern unter an, um die Optionen der aktuellen Konfiguration in eine Datei zu speichern, die als benannte Optionsgruppe geladen werden kann.
Laden...	Zeigt das Dialogfeld Optionsgruppe an, in dem Sie die Optionen einer benannten Optionsgruppe für die aktuelle Konfiguration übernehmen können.

Anmerkung: Es gibt einige Kompatibilitätoptionen, deren Schalter mit **-Vb** beginnen: **-Vbe**, **-Vbn**, **-Vbo**, **-Vbr**, **-Vbs** und **-Vbx**. Sie können alle mit dem Schalter **-Vb** gesetzt und zurückgesetzt werden: **-Vb** und **-Vb+** aktiviert alle **-Vb**-Schalter, **-Vb-** deaktiviert alle **-Vb**-Schalter und **-Vb.** setzt alle **-Vb**-Schalter auf ihre Standardwerte.

Allgemeine Kompatibilitätoptionen	Beschreibung
Nicht-konstante String-Literale (-Vbs)	Behandelt String-Literale nicht als 'const'. Vorgabe = nicht aktiviert
Globale Funktionen in Segmenten (-VA)*	Erzeugt alle globalen Funktionen in ihrem eigenen virtuellen oder schwachen Segment. Vorgabe = aktiviert
Aufrufkonvention nicht verkürzen (-VC)	Ist diese Option gesetzt, deaktiviert der Compiler die Unterscheidung von Funktionsnamen, wenn der einzige mögliche Unterschied in inkompatiblen Codeerzeugungsoptionen besteht. Wenn Sie diese Option aktivieren, kann der Linker z.B. nicht feststellen, ob ein Aufruf einer __fastcall -Elementfunktion mit der Aufrufkonvention cdecl erfolgt. Diese Einstellung ermöglicht das Compilieren von alten Bibliotheksdateien, die auf andere Weise nicht neu compiliert werden können. Vorgabe = nicht aktiviert
Microsoft-Header-Suchalgorithmus (-VI)*	Verwendet den Microsoft-Suchalgorithmus zum Suchen von Header-Dateien. Vorgabe = aktiviert
VC++-Kompatibilität (-VM)	Dient dem Herstellen von Kompatibilität mit Microsoft Visual C++; ersetzt die Aufrufkonvention __fastcall durch __msfastcall . Diese Option sollte nicht bei VCL-Anwendungen verwendet werden. Sie bewirkt zahlreiche Linker-Fehler. Vorgabe = nicht aktiviert
Lexikalischen Digraph-Scanner deaktivieren (-Vg)	Deaktiviert den lexikalischen Digraph-Scanner. Digraphen sind Sequenzen aus zwei Zeichen, die für ein einzelnes Zeichen stehen, das auf bestimmten Tastaturen nur schwer erzeugt werden kann. Wenn diese Option true ist, werden solche Diagraphen nicht erkannt. Vorgabe = nicht aktiviert

Neue Operatorennamen aktivieren (-Vn)	Ermöglicht neue Operatorennamen wie 'and', 'or', 'and_eq', 'bitand', etc. Vorgabe = nicht aktiviert
Alle Kompatibilitätsoptionen aktivieren (-Vo)	Setzt fast alle Kompatibilitäts-Flags, die in altem Code verwendet wurden; aktiviert -Vv , -Va, -Vp, -Vt, -Vc, -Vd und -Vx. Vorgabe = nicht aktiviert
Multizeichenkonstanten umkehren (-Vr)	Der Compiler dreht die Reihenfolge von Multizeichenkonstanten um. Vorgabe = nicht aktiviert
virdef-Generierung im alten Stil (-Vs)	Verwendet die virdef-Erzeugung im alten Stil. Vorgabe = nicht aktiviert
Native Quelltext für MBCS (-Vw)	Gibt nativen Code statt Unicode für Multibyte-Zeichen aus. Vorgabe = nicht aktiviert
Alte 8.3 Include-Suche (-Vi)	Verwendet den alten 8.3-Suchalgorithmus zum Suchen von Header-Dateien. Vorgabe = nicht aktiviert

Allgemeine Option	Beschreibung
Standard für neue Projekte	Speichert die aktuellen Einstellungen als Standardeinstellungen für neue Projekte.

2.225 C++-Compiler Allgemeine Compilierung

[Projekt](#)▸[Optionen](#)▸[C++-Compiler](#)▸[Allgemeine Compilierung](#)

Verwenden Sie dieses Dialogfeld dazu, allgemeine Compilieroptionen des C++-Compilers zu setzen.

Build-Konfigurationsoptionen	Beschreibung
Build-Konfiguration	Zeigt die aktive Build-Konfiguration an. Sie können das Drop-Down-Menü dazu verwenden, eine andere Build-Konfiguration auszuwählen.
Speichern unter...	Zeigt das Dialogfeld Speichern unter an, um die Optionen der aktuellen Konfiguration in eine Datei zu speichern, die als benannte Optionsgruppe geladen werden kann.
Laden...	Zeigt das Dialogfeld Optionsgruppe an, in dem Sie die Optionen einer benannten Optionsgruppe für die aktuelle Konfiguration übernehmen können.

Compilierungsoptionen	Beschreibung
Anweisungs-Set	<p>80386 (-3)* Erzeugt für den geschützten Modus eines 80386-Prozessors kompatible Anweisungen. Dies ist die Standardeinstellung.</p> <p>80486 (-4) Erzeugt für den geschützten Modus eines i486-Prozessors kompatible Anweisungen.</p> <p>Pentium (-5) Erzeugt Pentium-Instruktionen.</p> <p>Diese Option erhöht zwar die Ausführungsgeschwindigkeit auf Pentium-Rechnern, kann die Größe des Programms allerdings etwas erhöhen, wenn dieses mit den Optionen 80386 oder i486 compiliert wurde. Ferner ist bei für Pentium compiliertem Code mit einer Performance-Einbuße auf nicht-Pentium-Systemen zu rechnen.</p> <p>Pentium Pro (-6) Erzeugt Pentium Pro-Instruktionen.</p>

Datenausrichtung	<p>Byte (-a1)</p> <p>Forciert nicht die Ausrichtung von Variablen oder Datenfeldern an bestimmten Speichergrenzen. Der Compiler richtet die Daten an geraden oder ungeraden Adressen aus, je nach der nächsten verfügbaren Adresse.</p> <p>Während die Byte-Ausrichtung zwar kompaktere Programme erzeugt, sind diese meist jedoch etwas langsamer. Die anderen Optionen zur Datenausrichtung erhöhen die Geschwindigkeit, mit der 80x86-Prozessoren Daten holen und speichern.</p> <p>Word (-a2)</p> <p>2-Byte-Datenausrichtung. Richtet Nicht-Zeichendaten an geraden Adressen aus. Automatische und globale Variablen werden korrekt ausgerichtet. char- und unsigned char-Variablen und Felder können an jeder Adresse platziert werden; alle anderen werden an geraden Adressen ausgerichtet.</p> <p>Double Word (-a4)*</p> <p>4-Byte-Datenausrichtung. Richtet Nicht-Zeichendaten an 32-Bit-Grenzen aus (4 Byte). Datentypen mit weniger als vier Byte werden entsprechend ihrer Typgröße ausgerichtet. Dies ist die Standardeinstellung.</p> <p>Quad Word (-a8)</p> <p>8-Byte-Datenausrichtung. Richtet Nicht-Zeichendaten an 64-Bit-Grenzen aus (8 Byte). Datentypen mit weniger als acht Byte werden entsprechend ihrer Typgröße ausgerichtet.</p> <p>Paragraph (-a16)</p> <p>16-Byte-Datenausrichtung. Richtet Nicht-Zeichendaten an 128-Bit-Grenzen aus (16 Byte). Datentypen mit weniger als 16 Bytes werden entsprechend ihrer Typgröße ausgerichtet.</p>
Registervariablen	<p>Ohne (-r-)*</p> <p>Deaktiviert die Verwendung von Registervariablen. Teilt dem Compiler mit, keine Registervariablen zu verwenden, selbst dann nicht, wenn das Schlüsselwort register vorhanden ist. Dies ist die Standardeinstellung.</p> <p>Explizit (-rd)</p> <p>Verwendet Registervariablen nur, wenn Sie das Schlüsselwort register angeben und ein Register verfügbar ist. Mit dieser Option oder der Option Immer optimieren Sie die Nutzung von Registern. Sie können -rd in #pragma-Optionen verwenden.</p> <p>Immer (-r)</p> <p>Weist automatisch Registervariablen zu, wenn möglich, auch wenn Sie das Schlüsselwort register nicht verwenden.</p> <p>Im Allgemeinen können Sie die Einstellung für diese Option auf Immer belassen, es sei denn, Sie arbeiten mit bereits vorhandenem Assemblierungscode, der keine Registervariablen unterstützt.</p>

Aufrufkonvention	<p>Pascal (-p)</p> <p>Teilt dem Compiler mit, eine Pascal-Aufrufsequenz für Funktionsaufrufe (keine Unterstriche, nur Großbuchstaben, Aufruffunktion bereinigt den Stack, Parameter werden von links nach rechts verarbeitet) zu erzeugen. Diese Option hat denselben Effekt wie das Schlüsselwort _pascal. Die resultierenden Funktionsaufrufe sind normalerweise kleiner und laufen schneller als die, die mit der Aufrufkonvention C (-pc) generiert wurden. Funktionen müssen Argumente immer mit passendem Typ und passender Anzahl übergeben.</p> <p>Mit den Schlüsselwörtern _cdecl, _fastcall oder _stdcall kann eine Funktion oder eine Unterroutine explizit mit anderer Aufrufkonvention deklariert werden.</p> <p>C (-pc)*</p> <p>Teilt dem Compiler mit, eine C-Aufrufsequenz für Funktionsaufrufe (Unterstriche generieren, Berücksichtigung von Groß- und Kleinschreibung, Parameter werden von rechts nach links verarbeiten) zu erzeugen. Diese Option hat denselben Effekt wie das Schlüsselwort _cdecl. Funktionen, die mit der Aufrufkonvention C deklariert sind, können eine variable Parameterliste entgegennehmen, das heißt, die Anzahl der Parameter muss nicht fest sein.</p> <p>Mit den Schlüsselwörtern _pascal, _fastcall oder _stdcall kann eine Funktion oder eine Unterroutine explizit mit anderer Aufrufkonvention deklariert werden.</p> <p>Dies ist die Standardeinstellung.</p> <p>_msfastcall (-pm)</p> <p>Teilt dem Compiler mit, die Aufrufkonvention _msfastcall für alle Funktionen zu verwenden, für die keine explizite Aufrufkonvention deklariert ist.</p> <p>Fastcall (Register) (-pr)</p> <p>Zwingt den Compiler, alle Unterroutinen und Funktionen mit der Register-Parameterübergabekonvention zu erzeugen. Dies hat denselben Effekt wie das Deklarieren aller Unterroutinen und Funktionen mit dem Schlüsselwort _fastcall. Wenn diese Option aktiviert ist, erwarten Funktionen und Routinen die Parameterübergabe in Registern.</p> <p>Mit den Schlüsselwörtern _pascal, _cdecl oder _stdcall kann eine Funktion oder Unterroutine explizit mit anderer Aufrufkonvention deklariert werden.</p> <p>stdcall (-ps)</p> <p>Teilt dem Compiler mit, eine stdcall-Aufrufsequenz für Funktionsaufrufe (keine Unterstriche, Schreibweise beibehalten, Ablegen aufgerufener Funktionen auf dem Stack, Parameter von rechts nach links verarbeiten) zu erzeugen. Diese Option hat denselben Effekt wie das Schlüsselwort _stdcall. Funktionen müssen Argumente immer mit passendem Typ und passender Anzahl übergeben.</p> <p>Mit den Schlüsselwörtern _cdecl, _pascal oder _fastcall kann eine Funktion oder Unterroutine explizit mit anderer Aufrufkonvention deklariert werden.</p>
------------------	--

Komplianz	<p>ANSI (-A) Verwendet ANSI-Schlüsselwörter und -Erweiterungen. Es wird C- und C++-ANSI-kompatibler Quelltext kompiliert. Dies sorgt für maximale Portabilität. Nicht-ANSI-Schlüsselwörter werden nicht als Schlüsselwörter interpretiert.</p> <p>K & R (-AK) Verwendet Schlüsselwörter und Erweiterungen von Kernighan und Ritchie (K&R). Es werden nur die von Kernighan und Ritchie definierten Ergänzungen der Sprachschlüsselwörter erkannt. Alle C++-Erweiterungen von CodeGear werden als normale Bezeichner interpretiert.</p> <p>Borland (auch -A-) (-AT)* Verwendet Borland/CodeGear C++-Schlüsselwörter und -Erweiterungen. Weist den Compiler an, die CodeGear-Erweiterungen für die Schlüsselwörter der Programmiersprache C zu verwenden, einschließlich near, far, huge, asm, cdecl, pascal, interrupt, _export, _ds, _cs, _ss, _es und die Register-Pseudovariablen (_AX, _BX etc.). Dies ist die Standardeinstellung.</p> <p>Unix-System V (-AU) Verwendet Schlüsselwörter und Erweiterungen aus UNIX-System V. Es werden nur die Unix V Schlüsselwörter erkannt. Alle C++-Erweiterungsschlüsselwörter von CodeGear werden als normale Bezeichner interpretiert.</p> <p>Hinweis: Wenn im Quelltext Syntaxfehler bei Deklarationen gemeldet werden, sollten Sie prüfen, ob diese Option auf Borland-Erweiterungen gesetzt ist (Voreinstellung).</p>
Erweiterte Fehlerinformation (-Q)	Der Compiler erzeugt bei Fehlern umfassendere Informationen. (Vorgabe = nicht aktiviert)
Standard-Stack-Frames (-k)	<p>Erzeugt einen Standard-Stack-Frame (Standardcode für Einritt und Austritt in/aus der Funktion). Dies ist bei der Fehlersuche hilfreich, da sich der Stack der aufgerufenen Unterroutine einfacher durchlaufen lässt.</p> <p>Wenn diese Option deaktiviert ist, werden alle Funktionen, die keine lokalen Variablen verwenden und keine Parameter besitzen, mit verkürztem Eintritts- und Rückgabecode kompiliert. Damit wird der Code kleiner und schneller.</p> <p>Wenn eine Quelltextdatei zu Debugging-Zwecken kompiliert wird, sollte diese Option immer aktiviert sein.</p> <p>(Vorgabe = nicht aktiviert)</p>
Enums mit Integergröße (-b)	<p>Weist Aufzählungstypen (Variablen des Typs enum) ein ganzes Wort (ein 4 Byte Int bei 32-Bit-Programmen) zu.</p> <p>Ist die Option nicht aktiv (-b-), weist der Compiler den kleinsten Integer zu, der die Aufzählungswerte aufnehmen kann: Der Compiler weist den Typ unsigned oder signed char zu, wenn die Werte der Aufzählung im Bereich zwischen 0 und 255 (Minimum) oder zwischen -128 und 127 (Maximum) liegen. Befinden sich die Werte der Aufzählung in einem nachfolgenden Bereich, weist er den Typ unsigned oder signed short zu:</p> <p>0..65535 oder -32768..32767.</p> <p>Der Compiler weist den Aufzählungswerten ein 4-Byte Int (32 Bit) zu, wenn ein Wert außerhalb der dieser Bereiche liegt.</p> <p>(Vorgabe = aktiviert)</p>
Warnungen wie Fehler behandeln (-w!)	Veranlasst den Compiler, Warnungen wie Fehler zu behandeln. (Vorgabe = nicht aktiviert)
C++-Compilierung erzwingen (-P)	Veranlasst den Compiler, alle Quelltextdateien als C++-Dateien, unabhängig von ihrer Dateinamenserweiterung, zu compilieren. (Vorgabe = nicht aktiviert)
Stapel-Compilierung	Aktiviert die Stapeldatei-Compilierung. (Vorgabe = nicht aktiviert)

Allgemeine Option	Beschreibung
Standard für neue Projekte	Speichert die aktuellen Einstellungen als Standardeinstellungen für neue Projekte.

2.226 C++-Compiler

Projekt▸**Optionen**▸**C++-Compiler**

Dies ist der oberste Knoten für die Befehlszeilenoptionen des C++-Compilers.

Anmerkung: Optionen, die auf den Optionsseiten mit einem Sternchen (*) markiert sind, sind die Vorgabewerte.

Build-Konfigurationsoptionen	Beschreibung
Build-Konfiguration	Zeigt die aktive Build-Konfiguration an. Wählen Sie mit Hilfe der Drop-Down-Liste eine andere Build-Konfiguration aus.
Speichern unter...	Zeigt das Dialogfeld Speichern unter an, um die Optionen der aktuellen Konfiguration in eine Datei zu speichern, die als benannte Optionsgruppe geladen werden kann.
Laden...	Zeigt das Dialogfeld Optionsgruppe an, in dem Sie die Optionen einer benannten Optionsgruppe für die aktuelle Konfiguration übernehmen können.

Allgemeine Option	Beschreibung
Standard für neue Projekte	Speichert die aktuellen Einstellungen als Standardeinstellungen für neue Projekte.

Siehe auch

Allgemeine Compilierung (siehe Seite 538)

Erweiterte Compilierung (siehe Seite 526)

C++-Compilierung (siehe Seite 531)

Allgemeine Kompatibilität (siehe Seite 536)

C++-Kompatibilität (siehe Seite 529)

Debugging (siehe Seite 534)

Ausgabe (siehe Seite 545)

Optimierungen (siehe Seite 544)

Pfade und Definitionen (siehe Seite 546)

Vorcompilierte Header (siehe Seite 547)

Warnungen (siehe Seite 549)

2.227 C++-Compiler Optimierungen

Projekt▸**Optionen**▸**C++-Compiler**▸**Optimierungen**

Verwenden Sie dieses Dialogfeld dazu, Optimierungsoptionen des C++ Compilers zu setzen.

Build-Konfigurationsoptionen	Beschreibung
Build-Konfiguration	Zeigt die aktive Build-Konfiguration an. Wählen Sie mit Hilfe der Drop-Down-Liste eine andere Build-Konfiguration aus.
Speichern unter...	Zeigt das Dialogfeld Speichern unter an, um die Optionen der aktuellen Konfiguration in eine Datei zu speichern, die als benannte Optionsgruppe geladen werden kann.
Laden...	Zeigt das Dialogfeld Optionsgruppe an, in dem Sie die Optionen einer benannten Optionsgruppe für die aktuelle Konfiguration übernehmen können.

Optimierungsoptionen	Beschreibung
Keine (-Od)*	<p>Deaktiviert alle Optimierungseinstellungen, einschließlich derjenigen, die sie ausdrücklich eingestellt haben und derjenigen, die normalerweise als Teil des Geschwindigkeits-/Größekompromisses ausgeführt werden.</p> <p>Weil dadurch Kompaktheits- und Querverweisoptimierungen verhindert werden, kann diese Option den Debugger davon abhalten, herumzuspringen oder aus einer Funktion ohne Warnung zurückzukehren. Dadurch wird das schrittweise Abarbeiten des Codes erleichtert.</p> <p>Dies ist die Standardeinstellung.</p>
Größe (-O1)	Setzt Optimierungsoptionen, die den Compiler dazu veranlassen den Code hinsichtlich der Größe zu optimieren. Zum Beispiel durchsucht der Compiler den erzeugten Code nach doppelten Sequenzen. Wenn solche Sequenzen gefunden werden, ersetzt der Optimierungsvorgang eine Codesequenz durch einen Sprung auf die andere Sequenz und löscht den ersten Code-Teil. Dies ist häufig bei switch-Anweisungen der Fall. Der Compiler führt die Optimierung nach der Größe durch, indem er die kleinstmögliche Codesequenz auswählt.
Geschwindigkeit (-O2)	Setzt Optimierungsoptionen, die den Compiler dazu veranlassen, den Code hinsichtlich der Geschwindigkeit zu optimieren.
Ausgewählte	<p>Ermöglicht es, bestimmte Optimierungen zu aktivieren.</p> <p>Klicken Sie auf Alles markieren, um alle Optimierungen in der Liste zu aktivieren.</p> <p>Klicken Sie auf Keine markieren, um alle Optimierungen in der Liste zu deaktivieren.</p> <p>Klicken Sie auf Vorgabe, um die Standardoptimierungen in der Liste zu aktivieren.</p>

Allgemeine Option	Beschreibung
Standard für neue Projekte	Speichert die aktuellen Einstellungen als Standardeinstellungen für neue Projekte.

2.228 C++ Compiler Ausgabe

Projekt ▶ Optionen ▶ C++-Compiler ▶ Ausgabe

Verwenden Sie dieses Dialogfeld dazu, die Ausgabeoptionen des C++ Compilers zu setzen.

Build-Konfigurationsoptionen	Beschreibung
Build-Konfiguration	Zeigt die aktive Build-Konfiguration an. Wählen Sie mit Hilfe der Drop-Down-Liste eine andere Build-Konfiguration aus.
Speichern unter...	Zeigt der Dialogfeld Speichern unter an, um die Optionen der aktuellen Konfiguration in eine Datei zu speichern, die als benannte Optionsgruppe geladen werden kann.
Laden...	Zeigt das Dialogfeld Optionsgruppe an, in dem Sie die Optionen einer benannten Optionsgruppe für die aktuelle Konfiguration übernehmen können.

.obj-Inhaltsoptionen	Beschreibung
Autoabhängigkeitsausgabe des Compilers deaktivieren (-X)	Deaktiviert die Autoabhängigkeitsausgabe des Compilers. Vorgabe = nicht aktiviert
System-Header Abhängigkeitsinfo ausschließen (-mm)	aus Ignoriert bei der Erzeugung von Abhängigkeitsinformationen System-Header-Dateien. Vorgabe = nicht aktiviert
Unterstriche für Symbolnamen erzeugen (-u)*	Der Compiler fügt vor jedem globalen Bezeichner automatisch einen Unterstrich (_) ein (z.B. vor Funktionen und globalen Variablen), ehe diese im Objektmodul gespeichert werden. Pascal-Bezeichner (jene die durch das Schlüsselwort pascal modifiziert wurden) werden in Großbuchstaben konvertiert; ihnen wird kein Unterstrich vorangestellt. Unterstriche für C und C++ sind optional, aber Sie sollten diese Option aktivieren, um beim Linken mit CodeGear C++-Bibliotheken Fehler zu vermeiden. Vorgabe = aktiviert
Exportierten Symbolen keinen Unterstrich voranstellen (-vu)	Stellt exportierten Symbolnamen keine Unterstrichzeichen voran. Vorgabe = nicht aktiviert
Browser-Informationen in .obj-Dateien einfügen (-R)	Fügt Browser-Informationen in generierte .OBJ-Dateien ein. Vorgabe = nicht aktiviert

Allgemeine Option	Beschreibung
Standard für neue Projekte	Speichert die aktuellen Einstellungen als Standardeinstellungen für neue Projekte.

2.229 C++-Compiler Pfade und Definitionen

Projekt ▶ Optionen ▶ C++-Compiler ▶ Pfade und Definitionen

Verwenden Sie dieses Dialogfeld dazu, Optionen für Pfade und Definitionen des C++-Compiler zu setzen.

Build-Konfigurationsoptionen	Beschreibung
Build-Konfiguration	Zeigt die aktive Build-Konfiguration an. Wählen Sie mit Hilfe der Drop-Down-Liste eine andere Build-Konfiguration aus.
Speichern unter...	Zeigt der Dialogfeld Speichern unter an, um die Optionen der aktuellen Konfiguration in eine Datei zu speichern, die als benannte Optionsgruppe geladen werden kann.
Laden...	Zeigt das Dialogfeld Optionsgruppe an, in dem Sie die Optionen einer benannten Optionsgruppe für die aktuelle Konfiguration übernehmen können.

Pfade und Definitionen	Beschreibung
Include-Pfad (-I)	Gibt das Laufwerk und/oder die Verzeichnisse an, die Include-Dateien des Programms enthalten. Standard-Include-Dateien sind jene, die Sie in Größer-Kleiner-Zeichen (<>) in einer #include-Anweisung (z.B. #include <MeineDatei>) angeben. Klicken Sie auf [...], um ein Dialogfeld zu öffnen, in dem Sie die Suchpfadliste bearbeiten können. Markieren Sie das Kontrollfeld Zusammenführen, damit die Pfade des unmittelbaren Vorfahren übernommen werden, obwohl diese Liste nicht eigentlich verändert wird.
Definition (-D)	Definiert den für den Nullstring angegebenen Bezeichnernamen. -Dname= <i>string</i> definiert den Namen für den String. Bei dieser Zuweisung darf <i>string</i> keine Leerzeichen oder Tabulatoren enthalten. Sie können auch mehrere #define-Optionen in der Befehlszeile definieren, indem Sie eine der folgenden Methoden verwenden: Mehrere Definitionen lassen sich nach einer einzelnen -D-Option einfügen, indem Sie diese durch einen Semikolon (;) trennen und Werte mit einem Ist-Gleich-Zeichen (=) zuweisen. Zum Beispiel: BCC32.EXE -Dxxx;yyy=1;zzz=NO MYFILE.C Fügen Sie mehrere -D-Optionen ein, indem Sie diese mit einem Leerzeichen trennen. Zum Beispiel: BCC32.EXE -Dxxx -Dyyy=1 -Dzzz=NO MYFILE.C Klicken Sie auf [...], um ein Dialogfeld zu öffnen, in dem Sie die Definitionen bearbeiten können. Markieren Sie das Kontrollfeld Zusammenführen, damit die Definitionen des unmittelbaren Vorfahren übernommen werden, obwohl diese Liste nicht eigentlich verändert wird.
.obj-Ausgabeverzeichnis (-n)	Setzt das Ausgabeverzeichnis auf den angegebenen Pfad. Klicken Sie auf [...], um nach einem Ordner zu suchen..
Ziel-Windows-Version	Konditionale Definitionen zielen auf die höchste Version der Windows API-Header-Dateien ab. Wählen Sie eine BS-Version aus der Dropdown-Liste aus. Weitere Informationen finden Sie unter http://msdn2.microsoft.com/en-us/library/aa383745.aspx . Die Vorgabe ist "Nicht festgelegt".

Allgemeine Option	Beschreibung
Standard für neue Projekte	Speichert die aktuellen Einstellungen als Standardeinstellungen für neue Projekte.

2.230 C++-Compiler Vorcompilierte Header

Projekt▸**Optionen**▸**C++-Compiler**▸**Vorcompilierte Header**

Verwenden Sie dieses Dialogfeld dazu, Optionen für vorcompilierte Header des C++ Compilers zu setzen.

Build-Konfigurationsoptionen	Beschreibung
Build-Konfiguration	Zeigt die aktive Build-Konfiguration an. Wählen Sie mit Hilfe der Drop-Down-Liste eine andere Build-Konfiguration aus.
Speichern unter...	Zeigt das Dialogfeld Speichern unter an, um die Optionen der aktuellen Konfiguration in eine Datei zu speichern, die als benannte Optionsgruppe geladen werden kann.
Laden...	Zeigt das Dialogfeld Optionsgruppe an, in dem Sie die Optionen einer benannten Optionsgruppe für die aktuelle Konfiguration übernehmen können.

Verwendungsoptionen	Beschreibung
Verwendung	<p>Nicht verwenden Keine vorcompilierten Header verwenden.</p> <p>Erzeugen und verwenden (-H)* Die IDE erzeugt und verwendet vorcompilierte Header. Der Standarddateiname ist <i><Projektnname>.CSM</i> für IDE-Projekte und <i>BC32DEF.CSM</i> für Befehlszeilen-Compilierungen. Dies ist die Standardeinstellung.</p> <p>Verwenden, aber nicht erzeugen (-Hu) Compiler verwenden bereits vorhandene vorcompilierte Header-Dateien; neue vorcompilierte Header-Dateien werden nicht erzeugt.</p>
VCH-Dateiname (-H=)	<p>Legt den Namen Ihrer vorcompilierten Header-Datei fest. Der Compiler setzt den Namen des vorcompilierten Header auf den angegebenen Dateinamen. Klicken Sie auf [...], um ein Dialogfeld zur Dateiauswahl anzuzeigen.</p> <p>Wenn diese Option gesetzt ist, generiert der Compiler die angegebene vorcompilierte Header-Datei und verwendet diese.</p>
Vorcompilierte Header zwischenspeichern (-Hc)	Der Compiler speichert die erzeugten vorcompilierten Header im Zwischenspeicher. Dies ist besonders dann hilfreich, wenn mehr als eine Header-Datei vorcompiliert wird. Vorgabe = nicht aktiviert
Intelligent zwischengespeicherte vorcompilierte Header aktivieren (-Hs)*	Der Compiler speichert die vorcompilierten Header intelligent im Zwischenspeicher (beansprucht weniger Speicherplatz als die normale Zwischenspeicheroption -Hc). Das Zwischenspeichern von Header-Dateien ist sinnvoll, wenn Sie mehr als eine Header-Datei vorcompilieren. Vorgabe = aktiviert

Erzeugungsoptionen	Beschreibung
Header-Namen ersetzen: (-Hr)	Ersetzt den Header-Namen <i>name1</i> mit <i>name2</i> Klicken Sie auf [...], um ein Dialogfeld zu öffnen, in dem Sie die Liste der Header-Dateien verwalten können.

Vorcompilierung anhalten nach: (-Hh=)	<p>Beendet die Compilierung des vorcompilierten Header nach der Verarbeitung der angegebenen Datei. Mit dieser Option reduzieren Sie den Platz auf dem Datenträger, der für vorcompilierte Header erforderlich ist. Klicken Sie auf [...], um ein Dialogfeld zur Dateiauswahl anzuzeigen.</p> <p>Wenn Sie diese Option verwenden, muss die angegebene Datei von einer Quelltextdatei (include) werden, damit der Compiler eine .CSM-Datei generieren kann.</p> <p>Innerhalb Ihrer .CPP-Dateien können Sie auch #pragma hdrstop verwenden, um anzugeben, wann die Erzeugung von vorcompilierten Headern beendet werden soll.</p> <p>Sie können keinen Namen einer Header-Datei angeben, die von einer anderen Header-Datei einbezogen wird. Dies gilt beispielsweise für einen Header, der von WINDOWS.H einbezogen wird, denn das würde bewirken, dass die vorcompilierte Header-Datei bereits geschlossen würde, bevor die Compilierung von WINDOWS.H abgeschlossen ist.</p>
Header-Inhalt einbeziehen (-Hi)	Bezieht den Inhalt der angegebenen Header-Datei(en) ein. Klicken Sie auf [...], um ein Dialogfeld zu öffnen, in dem Sie die Liste der Header-Dateien verwalten können. Markieren Sie das Kontrollfeld Zusammenführen, damit die Dateien des unmittelbaren Vorfahren übernommen werden, obwohl diese Liste nicht eigentlich verändert wird.
VCH mit externen Typdateien aktivieren (-He)*	<p>Der Compiler generiert eine oder mehrere Dateien, die Debug-Typinformationen für alle Symbole im vorcompilierten Header enthalten. Die Dateien enden mit der Erweiterung .#xx, wobei xx für die erste generierte Datei 00 ist, während sich die Zahl für alle weiteren, erforderlichen Typinformationsdateien jeweils um 1 erhöht.</p> <p>Durch diese Option wird die Größe der .OBJ-Dateien deutlich reduziert, da die Debug-Typinformationen zentralisiert werden und sich nicht mehr in jeder .OBJ-Datei wiederholen.</p> <p>Vorgabe = aktiviert</p>

Allgemeine Option	Beschreibung
Standard für neue Projekte	Speichert die aktuellen Einstellungen als Standardeinstellungen für neue Projekte.

2.231 C++-Compiler Warnungen

[Projekt](#)▸[Optionen](#)▸[C++-Compiler](#)▸[Warnungen](#)

Verwenden Sie dieses Dialogfeld dazu, Warnungsoptionen des C++ Compiler zu setzen.

Build-Konfigurationsoptionen	Beschreibung
Build-Konfiguration	Zeigt die aktive Build-Konfiguration an. Wählen Sie mit Hilfe der Drop-Down-Liste eine andere Build-Konfiguration aus.
Speichern unter...	Zeigt der Dialogfeld Speichern unter an, um die Optionen der aktuellen Konfiguration in eine Datei zu speichern, die als benannte Optionsgruppe geladen werden kann.
Laden...	Zeigt das Dialogfeld Optionsgruppe anwenden an, in dem Sie die Optionen einer benannten Optionsgruppe für die aktuelle Konfiguration übernehmen können.

Warnungsoptionen	Beschreibung
Alle einschalten (-w)	Zeigt alle Warnungen und Fehlermeldungen an.
Alle ausschalten (-w-)	Zeigt keine Warnungen und Fehlermeldungen an.
Ausgewählte *	Ermöglicht es, bestimmte Warnungen zu aktivieren. Dies ist die Standardeinstellung. Klicken Sie auf Alles markieren, um alle Warnungen in der Liste anzuzeigen. Klicken Sie auf Keine markieren, um alle Warnungen in der Liste zu deaktivieren. Klicken Sie auf Vorgabe, um die Standard-Warnungen in der Liste anzuzeigen.

Allgemeine Option	Beschreibung
Standard für neue Projekte	Speichert die aktuellen Einstellungen als Standardeinstellungen für neue Projekte.

2.232 Ressourcen-Compiler

Projekt▸**Optionen**▸**Ressourcen-Compiler**

Dies ist der oberste Knoten für die Befehlszeilenoptionen des Ressourcen-Compilers.

Anmerkung: Optionen, die auf den Optionsseiten mit einem Sternchen (*) markiert sind, sind die Vorgabewerte.

Build-Konfigurationsoptionen	Beschreibung
Build-Konfiguration	Zeigt die aktive Build-Konfiguration an. Wählen Sie mit Hilfe der Drop-Down-Liste eine andere Build-Konfiguration aus.
Speichern unter...	Zeigt das Dialogfeld Speichern unter an, um die Optionen der aktuellen Konfiguration in eine Datei zu speichern, die als benannte Optionsgruppe geladen werden kann.
Laden...	Zeigt das Dialogfeld Optionsgruppe an, in dem Sie die Optionen einer benannten Optionsgruppe für die aktuelle Konfiguration übernehmen können.

Allgemeine Option	Beschreibung
Standard für neue Projekte	Speichert die aktuellen Einstellungen als Standardeinstellungen für neue Projekte.

Siehe auch

Pfade und Definitionen ( siehe Seite 552)

Optionen ( siehe Seite 551)

2.233 Ressourcen-Compiler Optionen

Projekt ▶ Optionen ▶ Ressourcen-Compiler ▶ Optionen

Verwenden Sie dieses Dialogfeld dazu, Optionen für den Ressourcen-Compiler zu setzen.

Build-Konfigurationsoptionen	Beschreibung
Build-Konfiguration	Zeigt die aktive Build-Konfiguration an. Wählen Sie mit Hilfe der Drop-Down-Liste eine andere Build-Konfiguration aus.
Speichern unter...	Zeigt das Dialogfeld Speichern unter an, um die Optionen der aktuellen Konfiguration in eine Datei zu speichern, die als benannte Optionsgruppe geladen werden kann.
Laden...	Zeigt das Dialogfeld Optionsgruppe an, in dem Sie die Optionen einer benannten Optionsgruppe für die aktuelle Konfiguration übernehmen können.

Optionen	Beschreibung
Ressourcentyp	Windows 3.1 (-31) Erzeugt eine mit Windows 3.1 kompatible Ressourcendatei (.RES). 16 Bit (-16) Erzeugt eine 16 Bit-Ressource. 32 Bit (-32)* Erzeugt eine 32 Bit-Ressource. Dies ist die Standardeinstellung.
Standardsprache (-l)	Legt die Standardsprache fest. -1409 repräsentiert beispielsweise Englisch. Weitere Informationen über das Festlegen von Sprachbezeichnern finden Sie unter http://msdn2.microsoft.com/en-us/library/ms776324.aspx .
Codeseite (-c)	Verwendet die angegebene Codeseite zur Ressourcenübersetzung. Wenn nicht -c verwendet wird, wird die ANSI-Codeseite benutzt.
Weitere Optionen	Geben Sie weitere Optionen für den Ressourcen-Compiler ein.
INCLUDE ignorieren (-x)	Ignoriert die Umgebungsvariable INCLUDE. Vorgabe = nicht aktiviert
Verbose-Meldungen (-v)	Der Linker gibt detaillierte Informationsmeldungen aus. Vorgabe = nicht aktiviert
Unterstützung für Multibyte-Zeichen (-m)	Unterstützt Multibyte-Zeichen. Vorgabe = nicht aktiviert

Allgemeine Option	Beschreibung
Standard für neue Projekte	Speichert die aktuellen Einstellungen als Standardeinstellungen für neue Projekte.

2.234 Ressourcen-Compiler Pfade und Definitionen

[Projekt](#)▶ [Optionen](#)▶ [Ressourcen-Compiler](#)▶ [Pfade und Definitionen](#)

Verwenden Sie dieses Dialogfeld dazu, Optionen für Pfade und Definitionen des Ressourcen-Compilers zu setzen.

Build-Konfigurationsoptionen	Beschreibung
Build-Konfiguration	Zeigt die aktive Build-Konfiguration an. Wählen Sie mit Hilfe der Drop-Down-Liste eine andere Build-Konfiguration aus.
Speichern unter...	Zeigt der Dialogfeld Speichern unter an, um die Optionen der aktuellen Konfiguration in eine Datei zu speichern, die als benannte Optionsgruppe geladen werden kann.
Laden...	Zeigt das Dialogfeld Optionsgruppe anwenden an, in dem Sie die Optionen einer benannten Optionsgruppe für die aktuelle Konfiguration übernehmen können.

Pfade und Definitionen	Beschreibung
Include-Pfad: (-I)	Gibt das Laufwerk und/oder die Verzeichnisse an, die Include-Dateien des Programms enthalten. Standard-Include-Dateien sind jene, die Sie in Größer-Kleiner-Zeichen (<>) in einer #include-Anweisung (z.B. #include <MeineDatei>) angeben. Klicken Sie auf [...], um das Dialogfeld Dateisuchpfad einbeziehen zu öffnen, in dem Sie die Pfadliste verwalten können. Markieren Sie das Kontrollfeld Zusammenführen, damit die Pfade des unmittelbaren Vorfahren übernommen werden, obwohl diese Liste nicht eigentlich verändert wird.
Definition: (-d)	Definiert eine Liste mit Präprozessorsymbolen. Klicken Sie auf [...], um das Dialogfeld Definiert ein Präprozessorsymbol zu öffnen, in dem Sie die Präprozessorsymbole verwalten können. Markieren Sie das Kontrollfeld Zusammenführen, damit die Definitionen des unmittelbaren Vorfahren übernommen werden, obwohl diese Liste nicht eigentlich verändert wird.
.obj-Ausgabeverzeichnis	Definiert das Ausgabeverzeichnis für OBJ-Dateien. Klicken Sie auf [...], um ein Dialogfeld zur Verzeichnisauswahl anzuzeigen.

Allgemeine Option	Beschreibung
Standard für neue Projekte	Speichert die aktuellen Einstellungen als Standardeinstellungen für neue Projekte.

2.235 Delphi-Compiler Compilierung

[Projekt](#) ▶ [Optionen](#) ▶ [Delphi-Compiler](#) ▶ [Compilierung](#)

Verwenden Sie dieses Dialogfeld dazu, Compilierungsoptionen des Delphi-Compiler zu setzen.

Build-Konfigurationsoptionen	Beschreibung
Build-Konfiguration	Zeigt die aktive Build-Konfiguration an. Wählen Sie mit Hilfe der Drop-Down-Liste eine andere Build-Konfiguration aus.
Speichern unter...	Zeigt der Dialogfeld Speichern unter an, um die Optionen der aktuellen Konfiguration in eine Datei zu speichern, die als benannte Optionsgruppe geladen werden kann.
Laden...	Zeigt das Dialogfeld Optionsgruppe an, in dem Sie die Optionen einer benannten Optionsgruppe für die aktuelle Konfiguration übernehmen können.

Syntaxoptionen	Beschreibung
Strenge VAR-Strings-Prüfung (-\$V+)*	<p>Diese Option (entspricht der Direktive \$V) ist nur für Delphi-Quelltext von Bedeutung, in dem kurze Strings verwendet werden. Sie dient der Abwärtskompatibilität mit früheren Versionen von Delphi und Borland Pascal. Die Option steuert die Typprüfung für kurze Strings, die als Variablenparameter übergeben werden. Wenn die Option aktiviert ist (entspricht {\$V+}), wird eine strenge Typprüfung durchgeführt, d.h. formale und tatsächliche Parameter müssen denselben String-Typ haben. Bei deaktivierter Option (entspricht {\$V-}) kann eine Variable eines kurzen String-Typs selbst dann als Parameter verwendet werden, wenn die deklarierte Maximallänge nicht mit der des formalen Parameters übereinstimmt.</p> <p>Vorgabe = aktiviert</p>
Vollständige boolesche Auswertung (-\$B+)	<p>Schaltet zwischen zwei unterschiedlichen Modellen der Delphi-Quelltexterzeugung für die Booleschen Operatoren AND und OR um. Wenn die Option aktiviert ist (entspricht {\$B+}), erzeugt der Compiler Code für die vollständige Auswertung eines Booleschen Ausdrucks. Das bedeutet, dass jeder Operand eines Booleschen Ausdrucks, der mit den Operatoren AND und OR gebildet wird, garantiert ausgewertet wird, auch wenn das Ergebnis des gesamten Ausdrucks bereits feststeht. Bei deaktivierter Option (entspricht {\$B-}) generiert der Compiler Code für die Kurzschlussauswertung Boolescher Ausdrücke, d.h. die Auswertung wird beendet, sobald das Ergebnis des gesamten Ausdrucks feststeht (die Auswertung erfolgt immer von links nach rechts).</p> <p>Vorgabe = nicht aktiviert</p>
Erweiterte (-\$X+)*	<p>Syntax</p> <p>Dient nur der Abwärtskompatibilität. Verwenden Sie diese Option (entspricht {\$X-}) nicht in Ihren Delphi-Anwendungen. Mit dieser Option lässt sich die erweiterte Syntax von Delphi aktivieren oder deaktivieren:</p> <p>Funktionsanweisungen Im Modus {\$X+} lassen sich Funktionsaufrufe als Prozedurenaufrufe verwenden, d.h. das Ergebnis eines Funktionsaufrufs kann ignoriert werden, anstatt an eine andere Funktion übergeben oder in einer Operation bzw. Zuweisung verwendet zu werden. Im Allgemeinen werden die von einer Funktion ausgeführten Berechnungen durch das Funktionsergebnis repräsentiert, das nicht ignoriert werden sollte. Manchmal führen Funktionen aber lediglich eine bestimmte Operation durch (z. B. einer globalen Variablen einen Wert zuweisen) und geben keinen Wert zurück, der weiterverwendet werden kann.</p> <p>Result-Variable Wenn die Option aktiviert ist (entspricht {\$X+}), kann die vordefinierte Variable <i>Result</i> für den Rückgabewert der Funktion verwendet werden.</p> <p>Nullterminierte Strings Wenn diese Option aktiviert ist, können Delphi-Strings nullbasierten Zeichen-Arrays (array[0..X] of Char) zugewiesen werden, die mit den PChar-Typen kompatibel sind.</p> <p>Vorgabe = aktiviert</p>

Typisierter '@'-Operator (-\$T+)	<p>Steuert, welche Zeigertypen vom Operator @ generiert werden, und steuert deren Kompatibilität. Bei deaktivierter Option (entspricht {\$T-}) ist das Ergebnis des Operators @ immer ein Zeiger ohne Typ (Pointer), der mit allen übrigen Zeigertypen kompatibel ist. Wenn @ für einen Variablenverweis (entspricht {\$T+}) verwendet wird, ist das Ergebnis ein typisierter Zeiger, der nur mit <i>Pointer</i> und anderen Zeigern auf den Variablenotyp kompatibel ist. Bei deaktivierter Option sind festgelegte Zeigertypen (also nicht vom allgemeinen Typ <i>Pointer</i>) nicht kompatibel (auch wenn sie Zeiger auf denselben Typ sind). Bei aktiverter Option sind Zeiger auf denselben Typ kompatibel.</p> <p>Vorgabe = nicht aktiviert</p>	
Offene String-Parameter (-\$P+)*	<p>Ist nur für Quelltext von Bedeutung, der mit Huge-Strings compiliert wurde. Diese Option dient der Abwärtskompatibilität mit früheren Versionen von Delphi und Borland Pascal. Die Option (entspricht \$P) legt die Bedeutung von Variablenparametern fest, die mit dem Schlüsselwort string im Status {\$H-} deklariert wurden. Bei deaktivierter Option (entspricht {\$P-}) sind Variablenparameter, die mit dem Schlüsselwort string deklariert wurden, normale Variablenparameter. Dagegen werden sie im Status {\$P+} als OpenString-Parameter behandelt. Der Bezeichner OpenString kann unabhängig von der Einstellung der Direktive \$P immer zur Deklaration von offenen String-Parametern verwendet werden.</p> <p>Vorgabe = aktiviert</p>	
Standardmäßig Strings-\$H+)*	lange	<p>Nur für Delphi für Win32. Diese Option (entspricht die Direktive \$H) legt fest, welche Bedeutung das reservierte Wort string hat, wenn es ohne Zusatz in einer Typdeklaration steht. Der generische Typ string kann entweder einen langen, dynamisch zugewiesenen String (vom Typ <i>AnsiString</i>) oder einen kurzen, statisch zugewiesenen String (vom Typ <i>ShortString</i>) repräsentieren. Durch die Standardeinstellung wird der generische String-Typ als langer <i>AnsiString</i> definiert.</p> <p>Alle Objekte in der Komponentenbibliothek werden in diesem Status compiliert. Für neue Komponenten sollten ebenfalls lange Strings verwendet werden. Dasselbe gilt für Quelltext, der Daten aus String-Eigenschaften der Komponentenbibliothek übernimmt. Die deaktivierte Option (entspricht {\$H-}) ist sinnvoll, wenn auf Quelltext aus älteren Versionen von Delphi zugegriffen wird, in dem standardmäßig kurze Strings verwendet werden. Die Bedeutung von String-Typdefinitionen kann lokal überschrieben werden, um die Generierung kurzer Strings sicherzustellen. Außerdem können kurze String-Typen als string[255] oder ShortString deklariert werden. Diese Typen sind eindeutig und von der aktiveren Option unabhängig.</p> <p>Vorgabe = aktiviert</p>
Schreibbare strukturierte Konstanten (-\$J+)	<p>Legt fest, ob typisierte Konstanten geändert werden können. Wenn die Option aktiviert ist (entspricht {\$J+}), ist eine Änderung möglich. Typisierte Konstanten sind in diesem Fall mit initialisierten Variablen vergleichbar. Bei deaktivierter Option (entspricht {\$J-}) sind typisierte Konstanten tatsächlich konstant. Jeder Versuch, sie zu ändern, führt zu einer Fehlermeldung durch den Compiler. Als schreibbar werden typisierte Konstanten bezeichnet, die zur Laufzeit als Variablen verwendet und somit geändert werden können.</p> <p>Deshalb muss älterer Quelltext, der änderbare typisierte Konstanten enthält, mit aktiverter Option compiliert werden, während für neue Anwendungen initialisierte Variablen verwendet werden sollten und der Quelltext mit deaktivierter Option compiliert werden sollte.</p> <p>Vorgabe = nicht aktiviert</p>	

Optionen zur Fehlersuche	Beschreibung
Debug-Informationen (-\$D+)	<p>Aktiviert oder deaktiviert die Erzeugung von Debug-Informationen. Diese Informationen beinhalten für jede Prozedur eine Tabelle mit Zeilennummern, in der Adressen des Objektcodes als Zeilennummern im Quelltext dargestellt werden. Bei Units werden die Debug-Informationen in der Unit-Datei gemeinsam mit dem Objektcode der Unit aufgezeichnet. Durch die Debug-Informationen erhöht sich die Größe der Unit-Datei. Das Compilieren von Programmen, die diese Unit verwenden, erfordert deshalb mehr Speicher. Die Größe und die Ausführungsgeschwindigkeit des ausführbaren Programms werden aber nicht nachteilig beeinflusst. Wenn ein Programm oder eine Unit mit dieser Option (entspricht {\$D+}) compiliert wird, können Sie das betreffende Modul mit dem integrierten Debugger in Einzelschritten testen und Haltepunkte setzen. Mit den Optionen Vollständige Debug-Informationen und Map-Datei (auf der Seite Linker des Dialogfelds Projektoptionen) können nur vollständige Zeileninformationen für Module erzeugt werden, die mit aktivierter Option compiliert wurden. Diese Option wird normalerweise zusammen mit der Option Lokale Symbole (dem Schalter \$L) eingesetzt, der die Erzeugung von lokalen Symbolinformationen für das Debugging steuert.</p> <p>Vorgabe = nicht aktiviert</p>
Lokale Debug-Symbole (-\$L+)*	<p>Aktiviert oder deaktiviert die Erzeugung von Symbolinformationen. Lokale Symbolinformationen sind die Namen und Typen aller lokalen Variablen und Konstanten eines Moduls, also die Symbole im implementation-Abschnitt des Moduls und in seinen Prozeduren und Funktionen. Bei Units werden die lokalen Symbolinformationen zusammen mit dem Objektcode in der Unit-Datei gespeichert. Lokale Symbolinformationen vergrößern eine Unit-Datei. Das Compilieren von Programmen, die diese Unit verwenden, erfordert also mehr Speicher. Die Größe und die Ausführungsgeschwindigkeit der Programme werden aber nicht nachteilig beeinflusst. Wenn ein Programm oder eine Unit mit dieser Option (entspricht {\$L+}) compiliert wird, können Sie die lokalen Variablen des Moduls im integrierten Debugger überprüfen und ändern. Außerdem können die Aufrufe von Prozeduren und Funktionen des Moduls über die Option Ansicht►Aufruf-Stack überprüft werden. Mit den Optionen Mit TD32 Debug-Info und Map-Datei (auf der Seite Linker des Dialogfelds Projektoptionen) können nur Symbolinformationen für Module erzeugt werden, die mit aktivierter Option compiliert wurden. Diese Option wird normalerweise zusammen mit der Option Debug-Informationen eingesetzt, die die Erzeugung von Tabellen mit Zeilennummern zu Testzwecken steuert. Diese Option wird ignoriert, wenn der Compiler die Option Debug-Informationen deaktiviert hat.</p> <p>Vorgabe = aktiviert</p>
Assertion (-\$C+)*	<p>Aktiviert oder deaktiviert die Erzeugung von Assertions-Code in einer Quelltextdatei. Die Option ist standardmäßig aktiviert (entspricht {\$C+}). Gewöhnlich werden in der Auslieferungsversion eines Produkts zur Laufzeit keine Asserts verwendet. Entfernen Sie die Markierung von dieser Option, um Asserts zu deaktivieren. Vorgabe = aktiviert</p>
Symbolreferenzinfo	<p>Ohne</p> <p>Verwendet keine Referenzinfos.</p> <p>Nur Definitionen -\$DEFINITIONINFO ON*</p> <p>Dies ist die Standardeinstellung.</p> <p>Referenzinfo -\$REFERENCEINFO ON</p>

Laufzeitfehlerprüfungsoptionen	Beschreibung
Bereichsüberprüfung (-\$R+)	<p>Aktiviert oder deaktiviert die Erzeugung von Bereichsprüfungscode. Wenn die Option aktiviert ist (entspricht {\$R+}), werden alle Ausdrücke, die Arrays und Strings indizieren, dahingehend überprüft, ob sie sich innerhalb der festgelegten Grenzen befinden. Der gleichen Prüfung werden alle Zuweisungen an skalare Variablen und Teilbereichsvariablen unterzogen. Das Fehlschlagen der Bereichsprüfung führt zu einer <i>ERangeError</i>-Exception (bzw. zum Programmabbruch, wenn die Exception-Behandlung nicht aktiviert ist). Die Aktivierung der Bereichsprüfung vergrößert und verlangsamt Ihr Programm.</p> <p>Vorgabe = nicht aktiviert</p>

E/A-Prüfung (-\$I+)	Aktiviert oder deaktiviert die automatische Code-Generierung, die nach jedem Aufruf einer E/A-Prozedur das Ergebnis überprüft. Wenn eine E/A-Prozedur bei aktiverter Option ein Ergebnis ungleich 0 zurückgibt, führt dies zu einer <i>EInOutError</i> -Exception (bzw. zum Programmabbruch, wenn die Exception-Behandlung nicht aktiviert ist). Ist die Option deaktiviert, muss die E/A-Operation durch einen Aufruf von <i>IOResult</i> auf Fehler geprüft werden. Vorgabe = aktiviert
Überlaufprüfung (-\$Q+)	Steuert die Erzeugung von Code für die Überlaufprüfung. Wenn die Option aktiviert ist (entspricht {\$Q+}), werden bestimmte arithmetische Integer-Operationen (+, -, *, Abs, Sqr, Succ, Pred, Inc und Dec) auf einen Überlauf geprüft. Dazu wird ihnen zusätzlicher Code hinzugefügt, der sicherstellt, dass das Ergebnis innerhalb des unterstützten Bereichs liegt. Das Fehlschlagen der Überlaufprüfung führt zu einer <i>EIntOverflow</i> -Exception (bzw. zum Programmabbruch, wenn die Exception-Behandlung nicht aktiviert ist). Diese Option wird normalerweise zusammen mit der Option zur Bereichsüberprüfung (Schalter \$R) eingesetzt, die die Erzeugung von Bereichsprüfungscode aktiviert bzw. deaktiviert. Die Aktivierung der Überlausprüfung vergrößert und verlangsamt Ihr Programm. Vorgabe = nicht aktiviert

Optionen für die Quelltexterzeugung	Beschreibung
Optimierung (-\$O+)*	Steuert die Codeoptimierung. Wenn diese Option aktiviert ist (entspricht {\$O+}), führt der Compiler eine Reihe von Codeoptimierungen aus, wie z.B. Variablen in CPU-Register platzieren, allgemeine Unterausdrücke entfernen und Induktionsvariablen erzeugen. Bei deaktivierter Option (entspricht {\$O-}) werden diese Optimierungen nicht ausgeführt. Außer in bestimmten Testsituationen sollte die Codeoptimierung immer aktiviert sein. Die Optimierungen des Delphi-Compilers führen zu keinerlei Änderungen der Funktionsweise des Programms. Mit anderen Worten: Der Compiler führt keine "unsicheren" Optimierungen durch, die die besondere Aufmerksamkeit des Programmierers erfordern. Mit dieser Option lassen sich Optimierungen nur für eine gesamte Prozedur oder Funktion ein- oder ausschalten. Dies ist nicht für bestimmte Zeilen innerhalb einer Routine möglich. Vorgabe = aktiviert
Stack-Frames erzeugen (-\$W+)	Nur für Delphi für Win32. Steuert die Erzeugung von Stack-Frames für Prozeduren und Funktionen. Wenn die Option aktiviert ist (entspricht {\$W+}), werden Stack-Frames für Prozeduren und Funktionen auch dann erzeugt, wenn sie nicht benötigt werden. Bei deaktivierter Option (entspricht {\$W-}), werden Stack-Frames nur generiert, wenn die Verwendung lokaler Variablen durch die Routine dies erforderlich macht. Bestimmte Debugger-Tools verlangen eine Erzeugung von Stack-Frames für alle Prozeduren und Funktionen. Sie sollten diese Option nur für diese Fälle aktivieren. Vorgabe = nicht aktiviert

Pentium(tm)-sichere Division (-\$U+)	<p>Nur für Delphi für Win32. Steuert die Erzeugung von Gleitkommacode, der vor der fehlerhaften FDIV-Anweisung schützt, die in bestimmten (frühen) Pentium-Prozessoren enthalten ist. Windows 95, Windows NT 3.51 und spätere Windows-BS-Versionen enthalten Code, der den Pentium-FDIV-Fehler korrigiert und das System schützt. Wenn die Option aktiviert ist (entspricht {\$U+}), werden alle Gleitkommadivisionen mit einer Laufzeitbibliotheksroutine ausgeführt. Beim ersten Aufruf einer Gleitkommadivision prüft die Routine, ob die FDIV-Anweisung des Prozessors korrekt arbeitet, und aktualisiert die in der Unit System deklarierte Variable <i>TestFDIV</i> entsprechend. Deren Wert bestimmt bei den folgenden Gleitkommadivisionen, wie vorzugehen ist.</p> <p>-1 bedeutet, dass die FDIV-Anweisung getestet und als fehlerhaft erkannt wurde.</p> <p>0 bedeutet, dass die FDIV-Anweisung noch nicht geprüft wurde.</p> <p>1 bedeutet, dass die FDIV-Anweisung getestet und als korrekt erkannt wurde.</p> <p>Bei Pentium-Prozessoren, die keine fehlerhafte FDIV-Anweisung enthalten, bewirkt die Aktivierung dieser Option nur eine geringfügige Leistungsminderung. Bei fehlerhaften Pentium-Prozessoren können Gleitkommadivisionen bei aktiverter Option bis zu dreimal länger dauern, liefern aber stets korrekte Ergebnisse. Bei deaktivierter Option (entspricht {\$U-}) werden Gleitkommadivisionen durch Inline-FDIV-Anweisungen ausgeführt. Sie garantieren optimale Ausführungsgeschwindigkeiten und Codegrößen, können auf fehlerhaften Pentium-Prozessoren jedoch falsche Ergebnisse liefern. Sie sollten daher die Option nur deaktivieren, wenn Sie sicher sind, dass der Code auf einem fehlerfreien Pentium-Prozessor läuft.</p> <p>Vorgabe = nicht aktiviert</p>
Record-Ausrichtung	<p>Aus (-\$A-)</p> <p>Felder werden entsprechend den Compiler-Vorgaben ausgerichtet.</p> <p>Byte (-\$A1)</p> <p>Felder werden nie ausgerichtet. Alle Records und Klassenstrukturen werden gepackt.</p> <p>Word (-\$A2)</p> <p>Felder in Record-Typen, die ohne den Modifizierer packed deklariert sind, und Felder in Klassenstrukturen werden an Wortgrenzen ausgerichtet.</p> <p>Double Word (-\$A4)</p> <p>Felder in Record-Typen, die ohne den Modifizierer packed deklariert sind, und Felder in Klassenstrukturen werden an Doppelwortgrenzen ausgerichtet.</p> <p>Quad Word (-\$A8)*</p> <p>Felder in Record-Typen, die ohne den Modifizierer packed deklariert sind, und Felder in Klassenstrukturen werden an Vierfachwortgrenzen ausgerichtet. Variablen und typisierte Konstanten werden unabhängig von der Direktive \$A immer für einen optimalen Zugriff ausgerichtet. Durch Setzen der Option auf \$A8 wird die Ausführung beschleunigt. Dies ist die Standardeinstellung.</p>
Mindestgröße für Enum	<p>Byte (-\$Z1)</p> <p>Die Minimumgröße für Enum ist 1 Byte.</p> <p>Word (-\$Z2)</p> <p>Die Minimumgröße für Enum ist 2 Byte.</p> <p>Double Word (-\$Z4)*</p> <p>Die Minimumgröße für Enum ist 4 Byte. Dies ist die Standardeinstellung.</p> <p>Quad Word (-\$Z8)</p> <p>Die Minimumgröße für Enum ist 8 Byte.</p>
Codeseite (—codepage)	<p>Geben Sie die Codeseite für die Sprache Ihrer Anwendung ein. Die Codeseite wird als Dezimalzahl angegeben, die eine bestimmte Zeichencodierungstabelle bezeichnet. Es gibt Standardwerte für zahlreiche Sprachen.</p> <p>Vorgabe = 0, keine Codeseite</p>

Allgemeine Option	Beschreibung
Standard für neue Projekte	Speichert die aktuellen Einstellungen als Standardeinstellungen für neue Projekte.

2.236 Delphi-Compiler

Projekt▸**Optionen**▸**Delphi-Compiler**

Dies ist der oberste Knoten für die Befehlszeilenoptionen des C++-Compilers.

Anmerkung: Optionen, die auf den Optionsseiten mit einem Sternchen (*) markiert sind, sind die Vorgabewerte.

--	--

Optionen, die auf den Optionsseiten mit einem Sternchen (*) markiert sind, sind die Vorgabewerte.

Build-Konfigurationsoptionen	Beschreibung
Build-Konfiguration	Zeigt die aktive Build-Konfiguration an. Wählen Sie mit Hilfe der Drop-Down-Liste eine andere Build-Konfiguration aus.
Speichern unter...	Zeigt das Dialogfeld Speichern unter an, um die Optionen der aktuellen Konfiguration in eine Datei zu speichern, die als benannte Optionsgruppe geladen werden kann.
Laden...	Zeigt das Dialogfeld Optionsgruppe anwenden an, in dem Sie die Optionen einer benannten Optionsgruppe für die aktuelle Konfiguration übernehmen können.

Allgemeine Option	Beschreibung
Standard für neue Projekte	Speichert die aktuellen Einstellungen als Standardeinstellungen für neue Projekte.

Siehe auch

Pfade und Definitionen (↗ siehe Seite 562)

Compilieren (↗ siehe Seite 553)

Weitere Optionen (↗ siehe Seite 560)

Warnungen (↗ siehe Seite 563)

2.237 Delphi-Compiler Weitere Optionen

Projekt ▶ Optionen ▶ Delphi-Compiler ▶ Weitere Optionen

Verwenden Sie dieses Dialogfeld dazu, weitere Optionen für den Delphi-Compiler zu setzen.

Build-Konfigurationsoptionen	Beschreibung
Build-Konfiguration	Zeigt die aktive Build-Konfiguration an. Wählen Sie mit Hilfe der Drop-Down-Liste eine andere Build-Konfiguration aus.
Speichern unter...	Zeigt das Dialogfeld Speichern unter an, um die Optionen der aktuellen Konfiguration in eine Datei zu speichern, die als benannte Optionsgruppe geladen werden kann.
Laden...	Zeigt das Dialogfeld Optionsgruppe an, in dem Sie die Optionen einer benannten Optionsgruppe für die aktuelle Konfiguration übernehmen können.

Weitere Optionen	Beschreibung
Unit-Aliase: (-A)	Setzt Unit-Aliase auf die angegebenen Aliase..
Diese Packages beim Compilieren verwenden: (-LU)	Linkt mit den angegebenen Packages dynamisch.
Weitere Optionen:	Weitere Optionen für die Übergabe an den Compiler.

C/C++-Ausgabeoptionen	Beschreibung
Objektdateien Header	<p>und Keine C/C++-Ausgabe*</p> <p>Es erfolgt keine Zwischenausgabe. Dies ist die Standardeinstellung.</p> <p>C .objs (-J)</p> <p>Erzeugt C .obj-Dateien.</p> <p>C++ .objs (-JP)</p> <p>Erzeugt C++ .obj-Dateien.</p> <p>C++ .objs, Header (-JPH)</p> <p>Erzeugt C++ .obj- und .hpp-Dateien.</p> <p>C++ .objs, Header, Namespaces (-JPHN)</p> <p>Erzeugt C++ .obj- und .hpp-Dateien, einschließlich Namespaces.</p> <p>C++ .objs, Header, Namespaces, Export (-JPHNE)</p> <p>Erzeugt C++ .obj- und .hpp-Dateien mit Namespaces und Exportsymbolen.</p> <p>C++ .objs, Namespaces (-JPN)</p> <p>Erzeugt C++ .obj- Dateien mit Namespaces.</p> <p>C++ .objs, Namespaces, Export (-JPNE)</p> <p>Erzeugt C++ .obj-Dateien mit Namespaces und Exportsymbolen.</p> <p>C++ .objs, Header, Export (-JPHE)</p> <p>Erzeugt C++ .obj- und .hpp-Dateien und Exportsymbole.</p> <p>C++ .objs, Export (-JPE)</p> <p>Erzeugt C++ .obj-Dateien und Exportsymbole.</p> <p>Alle C++Builder-Dateien erzeugen (auch Package-Libs)</p> <p>Erzeugt C++ .obj- und .hpp-Dateien mit Namespaces und Exportsymbolen und Packages</p>
Header-Dateiausgabe (-NH)	Definiert das Ausgabeverzeichnis für .hpp-Dateien. Klicken Sie auf die Ellipsen-Schaltfläche, um das Dialogfeld Ordner suchen zu öffnen.

Allgemeine Option	Beschreibung
Standard für neue Projekte	Speichert die aktuellen Einstellungen als Standardeinstellungen für neue Projekte.

2.238 Delphi-Compiler Pfade und Definitionen

Projekt ▶ Optionen ▶ Delphi-Compiler ▶ Pfade und Definitionen

Verwenden Sie dieses Dialogfeld dazu, die Pfad- und Definitionsoptionen des CodeGear Pascal Compilers zu setzen.

Build-Konfigurationsoptionen	Beschreibung
Build-Konfiguration	Zeigt die aktive Build-Konfiguration an. Wählen Sie mit Hilfe der Drop-Down-Liste eine andere Build-Konfiguration aus.
Speichern unter...	Zeigt das Dialogfeld Speichern unter an, um die Optionen der aktuellen Konfiguration in eine Datei zu speichern, die als benannte Optionsgruppe geladen werden kann.
Laden...	Zeigt das Dialogfeld Optionsgruppe an, in dem Sie die Optionen einer benannten Optionsgruppe für die aktuelle Konfiguration übernehmen können.

Pfade und Definitionen	Beschreibung
Include-Pfad: (-I)	<p>Bezieht die angegebenen Suchpfade ein.</p> <p>Klicken Sie auf [...], um das Dialogfeld Include-Pfad zu öffnen, in dem Sie die Definitionen verwalten können. Markieren Sie das Kontrollfeld Zusammenführen, damit die Pfade des unmittelbaren Vorfahren übernommen werden, obwohl diese Liste nicht eigentlich verändert wird.</p>
Definition: (-D)	<p>Definiert ein bedingtes Symbol. Die Direktive -Dsymbol definiert ein Symbol. Das definierte Symbol kann von den Direktiven {\$IFDEF symbol} und {\$IFNDEF symbol} oder von DEFINED symbol im Bedingungsausdruck der Direktive {\$IFC cond-expr} verwendet werden.</p> <p>Sie können ein Symbol auch in der Quelltextdatei mit der Direktive {\$DEFINE symbol} definieren.</p> <p>Mehrere Definitionen nach einer einzelnen -D-Option werden jeweils durch ein Semikolon (;) getrennt. Zum Beispiel: <code>DCC32.EXE -Dxxx;yyy;zzz MYFILE.PAS</code></p> <p>Sie können auch mehrere -D-Optionen einfügen, indem Sie diese durch ein Leerzeichen trennen. Zum Beispiel: <code>DCC32.EXE -Dxxx -Dyyy -Dzzz MYFILE.PAS</code></p> <p>Klicken Sie auf [...], um das Dialogfeld Symbole definieren zu öffnen, in dem Sie die Definitionen verwalten können. Markieren Sie das Kontrollfeld Zusammenführen, damit die Definitionen des unmittelbaren Vorfahren übernommen werden, obwohl diese Liste nicht eigentlich verändert wird.</p>
.obj-Ausgabeverzeichnis (-NO)	Definiert das Ausgabeverzeichnis für OBJ-Dateien. Klicken Sie auf [...], um ein Dialogfeld zur Verzeichnisauswahl anzuzeigen.

Allgemeine Option	Beschreibung
Standard für neue Projekte	Speichert die aktuellen Einstellungen als Standardeinstellungen für neue Projekte.

2.239 Delphi-Compiler Warnungen

Projekt▸**Optionen**▸**Delphi-Compiler**▸**Warnungen**

Verwenden Sie dieses Dialogfeld dazu, um Warnungsoptionen für den Delphi-Compiler zu setzen.

Build-Konfigurationsoptionen	Beschreibung
Build-Konfiguration	Zeigt die aktive Build-Konfiguration an. Wählen Sie mit Hilfe der Drop-Down-Liste eine andere Build-Konfiguration aus.
Speichern unter...	Zeigt das Dialogfeld Speichern unter an, um die Optionen der aktuellen Konfiguration in eine Datei zu speichern, die als benannte Optionsgruppe geladen werden kann.
Laden...	Zeigt das Dialogfeld Optionsgruppe an, in dem Sie die Optionen einer benannten Optionsgruppe für die aktuelle Konfiguration übernehmen können.

Warnungsoptionen	Beschreibung
Hinweise (-H)*	Gibt Hinweise aus. Vorgabe = aktiviert
Warnungen (-W)*	Gibt Warnmeldungen aus. Vorgabe = aktiviert
Ausgewählte Warnungen	Wählen Sie bestimmte Hinweise und Warnungen aus, die aktiviert werden sollen. Klicken Sie auf Alles markieren, um alle Hinweise und Warnungen aus der Liste anzuzeigen. Klicken Sie auf Keine markieren, um alle Hinweise und Warnungen in der Liste zu deaktivieren. Klicken Sie auf Vorgabe, um die Standard-Hinweise und Warnungen aus der Liste anzuzeigen.

Allgemeine Option	Beschreibung
Standard für neue Projekte	Speichert die aktuellen Einstellungen als Standardeinstellungen für neue Projekte.

2.240 Option suchen

Projekt ▶ Optionen

Mit diesem Dialogfeld können Sie eine bestimmte Option für das ausgewählte Build-Tool suchen.

Element	Beschreibung
Suchen nach	Gibt die Suchkriterien an. Sie können nach einer Option suchen, indem Sie deren Beschreibung, z.B. Datenausrichtung eingeben oder die zugehörige Kommandozeilenoption, z.B. -a .
Optionsliste	Führt alle Kommandozeilenoptionen auf, die dem Suchtext entsprechen. Wenn Sie eine Option auswählen und auf OK klicken, wird die ausgewählte Option auf der Seite Projektoptionen angezeigt.

2.241 Linker Linken

Projekt▶Optionen▶Linker▶Linken

Verwenden Sie dieses Dialogfeld dazu, Optionen für das Linken zu setzen.

Build-Konfigurationsoptionen	Beschreibung
Build-Konfiguration	Zeigt die aktive Build-Konfiguration an. Wählen Sie mit Hilfe der Drop-Down-Liste eine andere Build-Konfiguration aus.
Speichern unter...	Zeigt das Dialogfeld Speichern unter an, um die Optionen der aktuellen Konfiguration in eine Datei zu speichern, die als benannte Optionsgruppe geladen werden kann.
Laden...	Zeigt das Dialogfeld Optionsgruppe an, in dem Sie die Optionen einer benannten Optionsgruppe für die aktuelle Konfiguration übernehmen können.

Link-Optionen	Beschreibung
Dynamische RTL	Steuert, ob die C RTL die Laufzeitbibliothek (cc3280.dll) statisch oder dynamisch linkt. Vorgabe = nicht aktiviert
Vollständige Debug-Informationen (-v)	Fügt diejenigen Informationen in die Ausgabedatei ein, die erforderlich sind, um Ihre Anwendungen mit dem integrierten Debugger von C++Builder oder dem Turbo Debugger zu debuggen. In der Befehlszeile bewirkt diese Option, dass der Linker die Debug-Informationen in der ausführbaren Datei für alle Objektmodule einfügt, die Debug-Informationen enthalten. Sie können mit den Optionen -v+ und -v- die Debug-Informationen selektiv aktivieren oder deaktivieren, d.h. auf einer Modul-zu-Modul-Basis (aber nicht in derselben Befehlszeile, in der -v verwendet wurde). Ein Beispiel: Der folgende Befehl enthält Debug-Informationen für die Module mod2 und mod3; nicht jedoch für mod1 und mod4: ILINK32 mod1 -v+ mod2 mod3 -v- mod4 Vorgabe = nicht aktiviert
Ausgabedateien beibehalten (-Gk)	Teilt dem Linker mit, die Ausgabedateien beizubehalten, die sonst bei Fehlern gelöscht würden. Der Linker wurde geändert, um seine Ausgabedatei (EXE/DLL) zu löschen, wenn beim Linken Fehler auftreten. Das alte Verhalten bestand darin, diese Dateien zu belassen, anstatt sie zu löschen. Vorgabe = nicht aktiviert
Maximale Fehleranzahl (-Enn)	Gibt die Anzahl der maximal zulässigen Fehler (nn) an, ehe das Linken abgebrochen wird. Vorgabe = 0
Package-Bibliothek erzeugen (-Gl)	Erzeugt eine statische Package-Bibliothek, die Quelltext aus allen .OBJs des Package enthält, damit diese gelinkt werden können. Vorgabe = nicht aktiviert
.drc-Datei erzeugen (-GD)	Erzeugt Delphi-kompatible .RC-Dateien (drc-Dateien). Die drc-Datei hat denselben Namen wie die ausführbare Datei und wird in dasselbe Verzeichnis wie diese ausgegeben. Vorgabe = nicht aktiviert
Importbibliothek erzeugen (-Gi)	Erzeugt eine Importbibliothek (nur für DLL- und Package-Projekte) Vorgabe = nicht aktiviert
Inkrementelles Linken deaktivieren (-Gn)	Unterdrückt die Erzeugung von Linker-Statusdateien durch das Deaktivieren des inkrementellen Linkens. Wenn Sie -Gn wählen, benötigen nachfolgende Link-Vorgänge genauso lange wie der erste. Vorgabe = nicht aktiviert

Erweiterte Optionen	Beschreibung
Image-Prüfsumme ermitteln (-Gz)	Berechnet die Prüfsumme des Ziels und nimmt das Ergebnis in den PE-Header des Ziels auf. Diese Information wird von NT Kernel-Treibern und System-DLLs verwendet. Vorgabe = nicht aktiviert
Schnelles TLS (-Gt)	Weist TLS (Thread-lokaler Speicher) von Windows, zu anstatt den Mechanismus zur gemeinsamen Nutzung von TLS zu verwenden. Vorgabe = nicht aktiviert
Ressourcen ersetzen (-Rr)	Fügt Ressourcen ein und/oder ersetzt diese, ohne die vorhandenen Ressourcen zu entfernen. Vorgabe = nicht aktiviert
Linken unter Berücksichtigung der Schreibweise (-c)	Unterscheidet zwischen Klein- und Großschreibung in public und externen Symbolen. Normalerweise sollte diese Option aktiviert sein, da sowohl C als auch C++ zwischen Groß- und Kleinschreibung unterscheiden. Vorgabe = aktiviert
Verbose (-r)	Setzt die ausführliche Linkoption rlink32, wodurch beim Ressourcen-Linken detaillierte Informationen ausgegeben werden. Vorgabe = nicht aktiviert
Vor dem Linken Status leeren (-C)	Entfernt vorhandene inkrementelle Linker-Statusdateien und erstellt diese Dateien dann neu, ehe das Linken fortgesetzt wird. Diese Option ermöglicht es Ihnen, die Statusdateien zu aktualisieren. Vorgabe = nicht aktiviert
Dateiausrichtung (-Af)	<p>Gibt die Seitenausrichtung für Code und Daten innerhalb der ausführbaren Datei an. Der Linker verwendet den Dateiausrichtungswert, wenn er die verschiedenen Objekte und Abschnitte (z.B. Code und Daten) in die Datei schreibt. Angenommen, Sie verwenden den Vorgabewert 0x200, so speichert der Linker den Abschnitt des Image in 512-Byte-Grenzen in der ausführbaren Datei.</p> <p>Wenn Sie diese Option verwenden, müssen Sie einen Wert für die Dateiausrichtung angeben, der eine Potenz von 2 ist, wobei der kleinste Wert 16 ist.</p> <p>Der veraltete Stil dieser Option (/A:dd) wird weiterhin aus Gründen der Rückwärtskompatibilität unterstützt. Mit dieser Option wird die Dezimalzahl dd mit der Potenz von 2 multipliziert, um den Wert der Dateiausrichtung zu berechnen.</p> <p>Die Befehlszeilenversion dieser Option (/Afxxxx) akzeptiert entweder dezimale oder hexadezimale Zahlen als Dateiausrichtungswert. Die Einstellung des Wertes ist 512 (0x200).</p> <p>Vorgabe = 0x200</p>
Objektausrichtung (-Ao)	<p>Der Linker verwendet den Objektausrichtungswert dazu, die virtuellen Adressen der verschiedenen Objekte und Abschnitte (z.B. Code und Daten) in Ihrer Anwendung zu ermitteln. Angenommen, Sie geben als Objektausrichtungswert 8192 an, so richtet der Linker die virtuellen Adressen der Abschnitte im Image an den 8192-Byte-Grenzen (0x2000) aus.</p> <p>Wenn Sie diese Option verwenden, müssen Sie einen Wert für die Objektausrichtung angeben, der eine Potenz von 2 ist, wobei der kleinste Wert 4096 (0x1000) (Standard) ist.</p> <p>Die Befehlszeilenversion dieser Option (/Ao) akzeptiert entweder dezimale oder hexadezimale Zahlen als Objektausrichtungswert.</p> <p>Vorgabe = 0x1000</p>
DLLs verzögert laden (-d)	<p>Das verzögerte Laden von DLLs ist sinnvoll bei DLLs, die häufig von einer Anwendung benutzt werden, kann aber zu Lasten des Startvorgangs gehen. DLLs, die verzögert geladen werden, werden erst dann geladen und initialisiert, wenn der Eintrittspunkt in der DLL tatsächlich aufgerufen wird. Es gibt eine begleitende RTL-Unterstützung für das verzögerte Laden von DLLs, die der Benutzer verwenden kann, um Fehler beim Laden zu behandeln und das verzögerte Ladesystem bei Bedarf zu ersetzen.</p> <p>Klicken Sie auf [...], um ein Dialogfeld zu öffnen, in dem Sie die Liste der DLLs verwalten können. Markieren Sie das Kontrollfeld Zusammenführen, damit die DLLs des unmittelbaren Vorfahren übernommen werden, obwohl diese Liste nicht eigentlich verändert wird.</p>
Weitere Optionen	Geben Sie hier weitere Linker-Optionen ein.

Allgemeine Option	Beschreibung
Standard für neue Projekte	Speichert die aktuellen Einstellungen als Standardeinstellungen für neue Projekte.

2.242 Linker

Projekt>Optionen>Linker

Dies ist der oberste Knoten für die Befehlszeilenoptionen des Linkers.

Anmerkung: Optionen, die auf den Optionsseiten mit einem Sternchen (*) markiert sind, sind die Vorgabewerte.

Build-Konfigurationsoptionen	Beschreibung
Build-Konfiguration	Zeigt die aktive Build-Konfiguration an. Wählen Sie mit Hilfe der Drop-Down-Liste eine andere Build-Konfiguration aus.
Speichern unter...	Zeigt das Dialogfeld Speichern unter an, um die Optionen der aktuellen Konfiguration in eine Datei zu speichern, die als benannte Optionsgruppe geladen werden kann.
Laden...	Zeigt das Dialogfeld Optionsgruppe an, in dem Sie die Optionen einer benannten Optionsgruppe für die aktuelle Konfiguration übernehmen können.

Allgemeine Option	Beschreibung
Standard für neue Projekte	Speichert die aktuellen Einstellungen als Standardeinstellungen für neue Projekte.

Anmerkung: Viele Linker-Optionen und ihre Schalter werden auf anderen Linker-Optionsseiten beschrieben. Die entsprechenden Links finden Sie unten. Informationen über Linker-Optionen erhalten Sie auch, wenn Sie ilink32 in einem Befehlsfenster eingeben:

-C Vor dem Linken Status leeren
 -wxxx Warnungssteuerung
 -Enn Maximale Fehleranzahl
 -r Ausführliches Linken
 -q Banner unterdrücken
 -c Linken unter Berücksichtigung der Schreibweise
 -v Vollständige Debug-Informationen
 -Gn Keine Statusdateien
 -Gi Importbibliothek erzeugen
 -GD .DRC-Datei erzeugen
 Map-Dateisteuerung:
 -M Zuordnung zu gekürzten Namen
 -m Map-Datei mit publics
 -s Detaillierte Segmentzuordnung
 -x Keine Zuordnung
 Pfade:
 -I Verzeichnis für Zwischenausgabe
 -L Suchpfade für Bibliothek festlegen
 -j Objektsuchpfade festlegen
 Image-Steuerung:
 -d Verzögertes Laden einer DLL
 -Af:nnnn Dateiausrichtung festlegen
 -Ao:nnnn Objektausrichtung festlegen
 -ax Anwendungstyp festlegen
 -b:xxxx Image-Basisadresse festlegen
 -Txx Ausgabedateityp festlegen

-H:xxxx Heap-Reservegröße festlegen
-Hc:xxxx Heap-Übergabegröße festlegen
-S:xxxx Stack-Reservegröße festlegen
-Sc:xxxx Stack-Übergabegröße festlegen
-Vd.d Windows-Version festlegen
-Dstring Image-Beschreibung setzen
-Vd.d Subsystem-Version festlegen
-Ud.d Image-Benutzerversion festlegen
-GC Image-Kommentar-String festlegen
-GF Image-Flags setzen
-Gl Statisches Package
-Gpd Nur-Entwurfszeit-Package
-Gpr Nur-Laufzeit-Package
-GS Abschnitts-Flags setzen
-Gt Schnelles TLS
-Gz Image-Prüfsumme ermitteln
-Rr Ressourcen ersetzen

Siehe auch

Linken ( siehe Seite 565)

Ausgabeoptionen ( siehe Seite 570)

Warnungen ( siehe Seite 575)

2.243 Linker Ausgabeoptionen

Projekt▸**Optionen**▸**Linker**▸**Ausgabeoptionen**

Verwenden Sie dieses Dialogfeld dazu, die Ausgabeeinstellungen des Linkers zu setzen.

Build-Konfigurationsoptionen	Beschreibung
Build-Konfiguration	Zeigt die aktive Build-Konfiguration an. Wählen Sie mit Hilfe der Drop-Down-Liste eine andere Build-Konfiguration aus.
Speichern unter...	Zeigt der Dialogfeld Speichern unter an, um die Optionen der aktuellen Konfiguration in eine Datei zu speichern, die als benannte Optionsgruppe geladen werden kann.
Laden...	Zeigt das Dialogfeld Optionsgruppe anwenden an, in dem Sie die Optionen einer benannten Optionsgruppe für die aktuelle Konfiguration übernehmen können.

Map-Dateioptionen	Beschreibung
Typ	<p>Map-Datei mit Segmenten *</p> <p>Bezieht nur Segmente in die Map-Datei ein. Tritt ein, wenn weder <code>-m</code>, <code>-s</code> noch <code>-x</code> angegeben ist. Dies ist die Standardeinstellung.</p> <p>Map-Datei mit publics (-m)</p> <p>Weist den Linker an, eine Map-Datei zu erstellen, die einen Überblick über die Anwendungssegmente sowie zwei Listings der public Symbole enthält.</p> <p>Die Segmentliste enthält pro Segment eine Zeile, zeigt die Segment-Startadresse, die Segmentlänge, den Segmentnamen und die Segmentklasse.</p> <p>Die public Symbole werden in zwei Listen eingeteilt, die erste zeigt die Symbole in alphabetischer Reihenfolge, die zweite zeigt sie in aufsteigender Adressfolge. Symbole mit absoluten Adressen sind mit Abs gekennzeichnet.</p> <p>Eine Liste mit public Symbolen ist für das Debuggen sinnvoll: Viele Debugger verwenden public Symbole, wodurch sich beim Debuggen eine Referenz auf die Symboladresse herstellen lässt.</p> <p>Detaillierte Segmentzuordnung (-s)</p> <p>Erstellt die umfangreichste Map-Datei mit einer detaillierten Auflistung der Segmente der Map-Datei, die mit der Option Publics (-m) erstellt wurde. Die detaillierte Liste der Segmente enthält die Segmentklasse, den Segmentnamen, die Segmentgruppe, das Segmentmodul und die Segment ACBP-Informationen. Wenn dasselbe Segment in mehreren Modulen erscheint, wird jedes Modul in einer eigenen Zeile aufgeführt.</p> <p>Das ACBP-Feld codiert die Attribute A (Ausrichtung), C (Kombination) und B (Big = Groß) in einem Satz von 4-Bit-Feldern, wie von Intel definiert. ILINK32 verwendet nur drei dieser Felder: A, C und B. Der ACBP-Wert in der Zuordnung wird in Hexadezimalwerten ausgegeben. Die folgenden Feldwerte müssen mit OR verbunden werden, damit der ACBP-Wert gedruckt wird.</p> <p>Feld Wert Beschreibung</p> <p>A (Ausrichtung) 00 Ein absolutes Segment</p> <p>20 Byte-ausgerichtetes Segment</p> <p>40 Word-ausgerichtetes Segment</p> <p>60 Paragraph-ausgerichtetes Segment</p> <p>80 Seite-ausgerichtetes Segment</p> <p>A0 Nicht benannter absoluter Teil des Speichers</p> <p>C (Kombination) 00 Nicht kombinierbar</p> <p>08 Public kombiniertes Segment</p> <p>B (groß) 00 Segment kleiner als 64 K</p> <p>02 Segment genau 64 K</p> <p>Mit den gesetzten Segmentoptionen werden public Symbole ohne Referenzen als bereit (idle) gekennzeichnet. Ein bereites Symbol ist ein public definiertes Symbol in einem Modul, das keine Referenz von einem EXTDEF-Record oder einem anderen Modul aus dem Link enthält. Beispielsweise zeigt dieses Fragment aus dem public Symbolabschnitt einer Map-Datei an, dass Symbol1 und Symbol3 von dem gelinkten Image nicht referenziert werden:</p> <pre>0002:00000874 Idle (Bereit) Symbol1 0002:00000CE4 Symbol2 0002:000000E7 Idle (Bereit) Symbol3</pre> <p>Keine Zuordnung erzeugen (-x)</p> <p>Deaktiviert die Erstellung der Map-Datei.</p> <p>Der Linker erzeugt standardmäßig eine Map-Datei mit folgenden allgemeinen Segmentinformationen: Liste der Segmente, Programmstartadresse und eventuelle Warn- und Fehlermeldungen, die während des Linkens erzeugt wurden. Für diese Einstellung gibt es keinen Schalter. Verwenden Sie die Option <code>-x</code>, um die Erstellung der Standard-Map-Datei zu unterdrücken.</p>

Namen kürzen (-M)	Gibt die C++ Bezeichner in verkürzter Form in der Map-Datei aus, nicht mit ihrem vollständigen Namen. Damit lässt sich feststellen, nach welchem Prinzip Namen verkürzt werden. Solche Namen werden von einigen Dienstprogrammen als Eingabe benötigt.
-------------------	--

2

Versionierungsoptionen	Beschreibung
BS-Version (-V)	<p>Gibt die Windows-Versions-ID an, unter der die Anwendung ausgeführt werden soll. Der Linker setzt das Feld für die Subsystemversion im Header der .EXE-Datei auf die Zahl, die Sie in diesem Eingabefeld angeben.</p> <p>Sie können die Versions-ID von Windows auch im Bereich SUBSYSTEM der Moduldefinitionsdatei (.DEF) angeben. Doch jede Versionseinstellung, die Sie in der IDE oder der Befehlszeile vornehmen, überschreibt die Einstellung in der DEF-Datei.</p> <p>Wenn Sie die Befehlszeilenoption -Vd.d verwenden, setzt der Linker die Windows-Versions-ID auf die Zahl, die in d.d angegeben wurde. Wenn Sie z.B. -V4.0 angeben, setzt der Linker das Subsystem-Versionsfeld im Header der EXE-Datei auf 4.0, wodurch eine Windows 95-Anwendung angegeben wird.</p>
Benutzer-Version (-U)	<p>Gibt die Versions-ID Ihrer ausführbaren Datei an. Der Linker setzt das Feld im Header der .EXE-Datei auf die Zahl, die Sie angeben.</p> <p>Wenn Sie die Befehlszeilenoption -Ud.d verwenden, setzt der Linker die Versions-ID der Anwendung auf die Zahl, die in d.d angegeben wurde. Wenn Sie z.B. -V4.0 angeben, setzt der Linker das Benutzer-Versionsfeld im Header der EXE-Datei auf 4.0.</p>

Image-Beschreibungsoption	Beschreibung
Image-Beschreibung (-D)	Speichert die angegebene Beschreibung im PE-Image.

Zwischenausgabeoptionen	Beschreibung
Zwischenausgabe	Weist den Linker an, die Zwischenausgabedateien in das angegebene Verzeichnis zu platzieren. Klicken Sie auf [...], um das Dialogfeld Ordner suchen zu öffnen.

PE-Dateioptionen	Beschreibung
Basisadresse (-B)	Legt die Image-Basisadresse fest. Behält die Adressverschiebungstabelle bei. Wert in Hex oder Dezimal für 0x200 oder 512 Byte-Grenzen. Vorgabewert = 0x00400000
Minimale Stack-Größe (-Sc)	<p>Gibt die Größe des übergebenen Stack in Hexadezimal oder Dezimalzahlen an. Der minimal zulässige Wert für dieses Feld ist 4 K (0x1000), und jeder angegebene Wert muss gleich oder kleiner der Einstellung für die reservierte Stack-Größe (-S) sein.</p> <p>Wenn Sie hier die übergebene Stack-Größe angeben, werden alle STACKSIZE-Einstellungen in der Moduldefinitionsdatei überschrieben.</p> <p>Vorgabewert = 0x00002000</p>
Maximale Stack-Größe (-S)	<p>Gibt die Größe des reservierten Stack in Hexadezimal oder Dezimalzahlen an. Der kleinste zulässige Wert für dieses Feld ist 4 K (0x1000).</p> <p>Wenn Sie hier die reservierte Stack-Größe angeben, werden alle STACKSIZE-Einstellungen in der Moduldefinitionsdatei überschrieben.</p> <p>Vorgabewert = 0x00100000</p>

Minimale Heap-Größe (-Hc)	<p>Gibt die Größe des übergebenen Heap in Hexadezimal oder Dezimalzahlen an. Der minimal zulässige Wert für dieses Feld ist 0, und jeder angegebene Wert muss gleich oder kleiner der Einstellung für die reservierte Heap-Größe (-H) sein.</p> <p>Wenn Sie hier die übergebene Heap-Größe angeben, werden alle HEAPSIZE-Einstellungen in der Moduldefinitionsdatei überschrieben.</p> <p>Vorgabewert = 0x00001000</p>
Maximale Heap-Größe (-H)	<p>Gibt die Größe des reservierten Heap in Hexadezimal oder Dezimalzahlen an. Der kleinste zulässige Wert für dieses Feld ist 0.</p> <p>Wenn Sie hier die reservierte Heap-Größe angeben, werden alle HEAPSIZE-Einstellungen in der Moduldefinitionsdatei überschrieben.</p> <p>Vorgabewert = 0x00100000</p>
Abschnitts-Flags (-GS)	<p>Mit dem Schalter -GS können Sie einem benannten Image-Abschnitt Flags hinzufügen.</p> <p>Der Schalter fügt die Flags den bereits vorhandenen Standard-Flags für einen bestimmten Abschnitt hinzu. Es gibt keine Möglichkeit die Standard-Flags aus einem Abschnitt zu entfernen.</p> <p>Mögliche Flags sind:</p> <ul style="list-style-type: none"> E - Ausführbar C - Enthält Code I - Enthält initialisierte Daten R - Abschnitt ist lesbar W - Abschnitt ist beschreibbar S - Abschnitt ist gemeinsam D - Abschnitt ist verworfen K - Abschnitt muss nicht in den Zwischenspeicher K - Abschnitt muss nicht auf Seite <p>Vorgabewert = Keine Flags</p>
Image-Flags (-GF)	<p>Mit dem Schalter GF können Sie mehrere Flags für das Image festlegen. Folgende Flags werden unterstützt:</p> <ul style="list-style-type: none"> -GF:SWAPNET -GF:SWAPCD -GF:UNIPROCESSOR -GF:LARGEADDRESSAWARE -GF:AGGRESSIVE <p>SWAPNET weist das Betriebssystem an, das Image in eine lokale Auslagerungsdatei zu kopieren und es von dort aus auszuführen, wenn sich das Image auf einem Netzwerklaufwerk befindet.</p> <p>SWAPCD weist das Betriebssystem an, das Image in eine lokale Auslagerungsdatei zu kopieren und es von dort aus auszuführen, wenn das Image sich auf einem entfernbaren Datenträger (z.B. einer CD, Diskette, USB-Speicher-Stick befindet).</p> <p>UNIPROCESSOR teilt dem Betriebssystem mit, dass diese Anwendung nicht auf einem Multiprozessor-System ausgeführt werden kann.</p> <p>LARGEADDRESSAWARE teilt dem Betriebssystem mit, dass die Anwendung Adressen lesen kann, die Größer als 4 G sind.</p> <p>AGGRESSIVE gestattet dem Betriebssystem, die Arbeitsumgebung einer Anwendung aggressiv anzupassen, wenn sich die Anwendung im Bereitschaftszustand befindet. Dies ist für Bildschirmschoner und andere Prozesse ideal, die sich soweit wie möglich abseits der Hauptprozesse befinden sollten.</p> <p>Vorgabewert = Keiner</p>

Allgemeine Option	Beschreibung
Standard für neue Projekte	Speichert die aktuellen Einstellungen als Standardeinstellungen für neue Projekte.

2.244 Linker Warnungen

Projekt▸**Optionen**▸**Linker**▸**Warnungen**

Verwenden Sie dieses Dialogfeld dazu, Warnungsoptionen des Linkers zu setzen.

Build-Konfigurationsoptionen	Beschreibung
Build-Konfiguration	Zeigt die aktive Build-Konfiguration an. Wählen Sie mit Hilfe der Drop-Down-Liste eine andere Build-Konfiguration aus.
Speichern unter...	Zeigt das Dialogfeld Speichern unter an, um die Optionen der aktuellen Konfiguration in eine Datei zu speichern, die als benannte Optionsgruppe geladen werden kann.
Laden...	Zeigt das Dialogfeld Optionsgruppe an, in dem Sie die Optionen einer benannten Optionsgruppe für die aktuelle Konfiguration übernehmen können.

Warnungsoptionen	Beschreibung
Alle einschalten (-w)	Zeigt alle Warnungen und Fehlermeldungen an.
Alle ausschalten (-w-)	Zeigt keine Warnungen und Fehlermeldungen an.
Ausgewählte	Ermöglicht es, bestimmte Warnungen zu aktivieren. Dies ist die Standardeinstellung. Klicken Sie auf Alles markieren, um alle Warnungen in der Liste anzuzeigen. Klicken Sie auf Keine markieren, um alle Warnungen in der Liste zu deaktivieren. Klicken Sie auf Vorgabe, um die Standard-Warnungen in der Liste anzuzeigen.

Allgemeine Option	Beschreibung
Standard für neue Projekte	Speichert die aktuellen Einstellungen als Standardeinstellungen für neue Projekte.

2.245 Bibliotheksverwaltung

Projekt ▶ Optionen ▶ TLib

Verwenden Sie dieses Dialogfeld dazu, Bibliotheksoptionen (TLib) zu setzen.

Anmerkung: Optionen, die auf den Optionsseiten mit einem Sternchen (*) markiert sind, sind die Vorgabewerte.

Build-Konfigurationsoptionen	Beschreibung
Build-Konfiguration	Zeigt die aktive Build-Konfiguration an. Wählen Sie mit Hilfe der Drop-Down-Liste eine andere Build-Konfiguration aus.
Speichern unter...	Zeigt der Dialogfeld Speichern unter an, um die Optionen der aktuellen Konfiguration in eine Datei zu speichern, die als benannte Optionsgruppe geladen werden kann.
Laden...	Zeigt das Dialogfeld Optionsgruppe an, in dem Sie die Optionen einer benannten Optionsgruppe für die aktuelle Konfiguration übernehmen können.

TLib-Optionen	Beschreibung
Dynamische RTL	Steuert, ob die C RTL die Laufzeitbibliothek (cc3280.dll) statisch oder dynamisch linkt. Vorgabe = nicht aktiviert
In Bibliothek Schreibweise beachten (/c)	Gibt bei Symbolen, die eine Berücksichtigung der Groß-/Kleinschreibung erfordern, eine Warnung aus. Vorgabe = nicht aktiviert
Erweitertes Verzeichnis erstellen (/E)	Erstellt ein erweitertes Verzeichnis. Vorgabe = nicht aktiviert
Ohne Kommentardatensätze (/0)	Entfernt Kommentardatensätze. Entfernt zusätzliche Records, wie etwa Zeilenummern. Vorgabe = nicht aktiviert
Seitengröße (/P)	Setzt die Größe für die Bibliotheksseite. Vorgabe = 0x0010
Listing-Dateiname	Legt den Namen der Listing-Datei fest. Klicken Sie auf [...], um ein Dialogfeld zur Dateiauswahl anzuzeigen.

Allgemeine Option	Beschreibung
Standard für neue Projekte	Speichert die aktuellen Einstellungen als Standardeinstellungen für neue Projekte.

2.246 Listeneditor

Projekt ▶ Optionen ▶ verschiedene Pfade

In diesem Dialogfeld können Sie eine Liste von Strings bearbeiten, die durch Semikolon getrennt sind.

Anmerkung: Nicht alle im Folgenden beschriebenen Optionen stehen für alle Projekttypen zur Verfügung.

Element	Beschreibung	
Stringliste	Führt alle jene Strings auf, die aktiv in der Build-Konfiguration gesetzt sind.	
Textfeld	Gibt den String an, der in die Stringliste eingefügt werden soll. Zeigt den aktuell ausgewählten String.	
[Pfeil auf]	Verschiebt den ausgewählten String in der Liste nach oben.	
[Pfeil ab]	Verschiebt den ausgewählten String in der Liste nach unten.	
Ellipsen-Schaltfläche	Zeigt das Dialogfeld Ordner suchen an. Verwenden Sie dieses Dialogfeld zum Erstellen eines Pfads.	
Ersetzen	Ersetzt den ausgewählten String durch den Inhalt im Textfeld.	
Hinzufügen	Fügt den String in das Textfeld der Liste ein.	
Löschen	Löscht den ausgewählten String.	
Ungültige löschen	Pfade	Löscht alle ungültigen Pfade aus der Stringliste.
Geerbte Werte	Führt die geerbten Strings aus Alle Konfigurationen auf. Sie können diese Liste nicht bearbeiten.	
Werte Einstellungen niedrigere erben	aus für Ebenen	Bewirkt, dass die aktive Build-Konfiguration Strings erbt, die in Alle Konfigurationen angegeben sind.

2.247 Pfade und Definitionen

Projekt>Optionen>Pfade und Definitionen

Verwenden Sie dieses Dialogfeld dazu, Pfade und Definitionen für das Projekt zu setzen.

Build-Konfigurationsoptionen	Beschreibung
Build-Konfiguration	Zeigt die aktive Build-Konfiguration an. Wählen Sie mit Hilfe der Drop-Down-Liste eine andere Build-Konfiguration aus.
Speichern unter...	Zeigt der Dialogfeld Speichern unter an, um die Optionen der aktuellen Konfiguration in eine Datei zu speichern, die als benannte Optionsgruppe geladen werden kann.
Laden...	Zeigt das Dialogfeld Optionsgruppe an, in dem Sie die Optionen einer benannten Optionsgruppe für die aktuelle Konfiguration übernehmen können.

Pfadoptionen	Beschreibung
Include-Pfad: (-I)	Legt die Verzeichnisse fest, die nach Include-Dateien durchsucht werden sollen. Diese ist eine Gruppe von Include-Pfaden, die an alle Tool-spezifischen Include-Pfade auf Projektebene angehängt werden. Standard-Include-Dateien sind jene, die Sie in Größer-Kleiner-Zeichen (<>) in einer #include-Anweisung (z.B. #include <MeineDatei>) angeben. Klicken Sie auf [...], um ein Dialogfeld zu öffnen, in dem Sie die Pfadliste bearbeiten können. Markieren Sie das Kontrollfeld Zusammenführen, damit die Pfade des unmittelbaren Vorfahren übernommen werden, obwohl diese Liste nicht eigentlich verändert wird.
Bibliothekspfad: (-L)	Legt die Verzeichnisse fest, die der Linker durchsucht, wenn für ein LIB-Modul in der Compilier/Link-Anweisung kein expliziter Pfad angegeben ist. Um den Bibliothekssuchpfad anzugeben, ist folgende Befehlszeilensyntax erforderlich: /L<Pfadangabe>[;<Pfadangabe>][. .] Der Linker verwendet die angegebenen Bibliothekssuchpfade, wenn kein expliziter Pfad für die LIB-Datei angegeben ist, und der Linker die Bibliotheksdatei nicht im aktuellen Verzeichnis finden kann. Der Befehl ILINK32 /Lc:\mylibs;.\libs splash.\common\logo,,,utils logolib beauftragt den Linker beispielsweise, zuerst das aktuelle Verzeichnis nach SPLASH.LIB zu durchsuchen. Wenn diese Datei im aktuellen Verzeichnis nicht gefunden wird, sucht der Linker im Verzeichnis C:\MYLIBS und dann im Verzeichnis .\LIBs. Beachten Sie jedoch, dass der Linker die Bibliothekssuchpfade nicht verwendet, um die Datei LOGO.LIB zu suchen, da dieser Datei ein expliziter Pfad zugewiesen wurde. Klicken Sie auf [...], um ein Dialogfeld zu öffnen, in dem Sie die Pfadliste verwalten können. Markieren Sie das Kontrollfeld Zusammenführen, damit die Pfade des unmittelbaren Vorfahren übernommen werden, obwohl diese Liste nicht eigentlich verändert wird.
Zwischenausgabe:	Weist den Linker an, die Zwischenausgabedateien in das angegebene Verzeichnis zu platzieren. Teilt außerdem den Compilern (dcc, bcc, tasm, brcc) mit, wo die compilierte Ausgabe (normalerweise .obj- und .rcs-Dateien) abgelegt werden soll. Aktuell werden die Linker-Statusdateien in diesem Verzeichnis abgelegt. Die .map-Datei und die .tds-Dateien werden in demselben Verzeichnis platziert wie das Ausgabe-Image, sofern nicht anders angegeben. Klicken Sie auf [...], um ein Dialogfeld zur Verzeichnisauswahl anzuzeigen.
Endgültige Ausgabe:	Gibt das Verzeichnis an, in dem die endgültige Ausgabe des Builds, wie z.B. die ausführbare Datei oder DLLs, platziert werden soll. Klicken Sie auf [...], um ein Dialogfeld zur Verzeichnisauswahl anzuzeigen.

BPI/Lib-Ausgabe: (-l)	Weist den Linker an, die bpi/lib-Ausgabedateien in das angegebene Verzeichnis zu legen. Klicken Sie auf [...], um ein Dialogfeld zur Verzeichnisauswahl anzuzeigen.
-----------------------	---

Definitionen	Beschreibung
Definitionen	<p>Definiert den für den Nullstring angegebenen Bezeichnernamen. <code>-Dname=string</code> definiert den Namen für den String. Bei dieser Zuweisung darf <code>string</code> keine Leerzeichen oder Tabulatoren enthalten. Sie können auch mehrere #define-Optionen in der Befehlszeile definieren, indem Sie eine der folgenden Methoden verwenden: Diese Option ist für das gesamte Projekt gültig: Diese Definitionen werden an die für spezielle Compiler angehängt.</p> <p>Mehrere Definitionen lassen sich nach einer einzelnen <code>-D</code>-Option einfügen, indem Sie diese durch einen Semikolon (<code>;</code>) trennen und Werte mit einem Ist-Gleich-Zeichen (<code>=</code>) zuweisen. Zum Beispiel: <code>BCC32.EXE -Dxxxx;yyy=1;zzz=NO MYFILE.C</code></p> <p>Fügen Sie mehrere <code>-D</code>-Optionen ein, indem Sie diese mit einem Leerzeichen trennen. Zum Beispiel: <code>BCC32.EXE -Dxxxx -Dyyy=1 -Dzzz=NO MYFILE.C</code></p> <p>Klicken Sie auf [...], um ein Dialogfeld zu öffnen, in dem Sie die Definitionen verwalten können. Markieren Sie das Kontrollfeld Zusammenführen, damit die Definitionen des unmittelbaren Vorfahren übernommen werden, obwohl diese Liste nicht eigentlich verändert wird.</p>

Allgemeine Option	Beschreibung
Standard für neue Projekte	Speichert die aktuellen Einstellungen als Standardeinstellungen für neue Projekte.

2.248 Projekteigenschaften

Projekt oder Tools ▶ Optionen ▶ Projekteigenschaften oder Umgebungsoptionen\C++-Optionen

Mit diesem Dialogfeld legen Sie Projekteigenschaften fest, die bestimmte Aspekte der Projektverwaltung in der IDE steuern.

Beachten Sie bitte, dass dieses Dialogfeld entweder über den Befehl **Projekt**▶**Optionen** oder über **Tools**▶**Optionen** angezeigt werden kann. Optionen, die über **Projekt**▶**Optionen** festgelegt wurden, gelten nur für diese Projekt. Optionen, die über **Tools**▶**Optionen** festgelegt wurden, gelten nur für neue Projekte.

C++-Projekteigenschaften	Beschreibung
Include- und Bibliothekspfade verwalten	Wenn diese Option markiert ist, werden die Pfade der Dateien, die ein Benutzer dem Projekt hinzufügt, der entsprechenden Include-Pfadoption hinzugefügt, um sicherzustellen, dass der Compiler/Linker diese Dateien findet. Ist diese Option nicht markiert, werden Include-Pfade nicht automatisch aktualisiert. Es obliegt dann dem Benutzer, sicherzustellen, dass die Include- und Bibliothekspfade korrekt sind. Vorgabe = aktiviert
Package-Importe und Bibliotheken verifizieren	Wenn diese Option markiert ist, wird vor dem Linken überprüft, ob alle Package-bezogenen Bibliotheken gefunden werden. Wenn eine Datei nicht gefunden wird, wird ein Dialogfeld angezeigt, in dem der Benutzer die Dateiposition angeben kann. Die Include-Pfade werden dann entsprechend aktualisiert. Ist diese Option nicht markiert, wird diese Überprüfung nicht ausgeführt. Vorgabe = aktiviert
Header-Abhängigkeiten in der Projektverwaltung anzeigen	Wenn diese Option markiert ist, wird in der Projektverwaltung eine Liste mit allen Header-Dateien erstellt und angezeigt, von denen eine C/C++-Datei abhängig ist (sofern diese Information zur Verfügung steht). Ist diese Option nicht markiert, wird die Liste nicht erstellt. Vorgabe = nicht aktiviert
Auto-Abhängigkeitsprüfung verwenden, falls verfügbar	Wenn für eine Quelltextdatei bereits eine Objektdatei vorhanden ist, erstellt ein Tool eine neue Objektdatei, sofern das Änderungsdatum der Quelltextdatei jünger als das der Objektdatei ist. Wenn diese Option markiert ist, erstellt das Tool eine neue Objektdatei, sofern das Änderungsdatum einer Include-Datei, von der die Quelltextdatei abhängt, neuer ist als das der Objektdatei. Ist diese Option nicht markiert, überprüft das Tool nicht alle Include-Dateien. Das Aktivieren dieser Option unterstützt akkurate Builds. Vorgabe = aktiviert
Allgemeine Meldungen anzeigen	Zeigt alle Meldungen ohne Filterung an. Vorgabe = nicht aktiviert

Allgemeine Option	Beschreibung
Standard für neue Projekte	Speichert die aktuellen Einstellungen als Standardeinstellungen für neue Projekte.

2.249 Turbo Assembler

Projekt▸**Optionen**▸**Tasm**

Dies ist der oberste Knoten für die Befehlszeilenoptionen des Turbo Assemblers.

Anmerkung: Optionen, die auf den Optionsseiten mit einem Sternchen (*) markiert sind, sind die Vorgabewerte.

Build-Konfigurationsoptionen	Beschreibung
Build-Konfiguration	Zeigt die aktive Build-Konfiguration an. Wählen Sie mit Hilfe der Drop-Down-Liste eine andere Build-Konfiguration aus.
Speichern unter...	Zeigt das Dialogfeld Speichern unter an, um die Optionen der aktuellen Konfiguration in eine Datei zu speichern, die als benannte Optionsgruppe geladen werden kann.
Laden...	Zeigt das Dialogfeld Optionsgruppe an, in dem Sie die Optionen einer benannten Optionsgruppe für die aktuelle Konfiguration übernehmen können.

Allgemeine Option	Beschreibung
Standard für neue Projekte	Speichert die aktuellen Einstellungen als Standardeinstellungen für neue Projekte.

Siehe auch

Pfade und Definitionen ( siehe Seite 585)

Optionen ( siehe Seite 582)

Warnungen ( siehe Seite 586)

2.250 Turbo Assembler Optionen

Projekt▸**Optionen**▸**Tasm**▸**Optionen**

Verwenden Sie dieses Dialogfeld zum Festlegen der Assembler-Optionen.

Build-Konfigurationsoptionen	Beschreibung
Build-Konfiguration	Zeigt die aktive Build-Konfiguration an. Wählen Sie mit Hilfe der Drop-Down-Liste eine andere Build-Konfiguration aus.
Speichern unter...	Zeigt das Dialogfeld Speichern unter an, um die Optionen der aktuellen Konfiguration in eine Datei zu speichern, die als benannte Optionsgruppe geladen werden kann.
Laden...	Zeigt das Dialogfeld Optionsgruppe an, in dem Sie die Optionen einer benannten Optionsgruppe für die aktuelle Konfiguration übernehmen können.

Optionen zur Fehlersuche	Beschreibung
Fehlersuche	<p>Vollständig (/zi) Ermöglicht die vollständige Nutzung der Funktionalität des Debuggers, um das Programm schrittweise zu durchlaufen oder Datenelemente zu ändern.</p> <p>Nur Zeilenummern (/zd)* Teilt dem Turbo Assembler mit, Zeilenummern einzufügen, damit eine Synchronisierung von Quelltextanzeige und Datentypinformationen möglich wird. Dies ist die Standardeinstellung.</p> <p>Keine (/zn) Deaktiviert Debug-Informationen in der Objektdatei.</p>

Optionen für die Quelltexterzeugung	Beschreibung
Overlay	<p>Standard (keine Overlays) (/os)* Erstellt Standardobjektcode ohne Overlays. Dies ist die Standardeinstellung.</p> <p>Standard (TLINK-Overlays) (/o) Erstellt Standardobjektcode mit TLINK-Overlays.</p> <p>Phar Lap-Korrekturen (/op) Erstellt Objektcode mit Phar Lap Overlay-kompatiblen Korrekturen.</p> <p>IBM-Korrekturen (/oi) Erstellt Objektcode mit IBM Overlay-kompatiblen Korrekturen.</p>
Segmentsortierung	<p>Alphabetisch (/a) Ordnet die Segment in alphabetischer Reihenfolge.</p> <p>Sequentiell (/s)* Ordnet die Segment in der Reihenfolge, in der Sie auftreten. Dies ist die Standardeinstellung.</p>

Gleitkomma	Emuliert (/e)* Aktiviert die emulierten Gleitkomma-Anweisungen. Dies ist die Standardeinstellung. Real (/r) Aktiviert die realen Gleitkomma-Anweisungen.
Groß-/Kleinschreibung	Groß-/Kleinschreibung nicht berücksichtigen (/mu)* Berücksichtigt die Schreibweise von Symbolnamen wird nicht. Dies ist die Standardeinstellung. Groß-/Kleinschreibung berücksichtigen (/ml) Behandelt alle Symbole innerhalb der Quelltextdatei so, als stünden sie in Großbuchstaben. Globale Symbole mit Berücksichtigung der Schreibweise (/mx) Berücksichtigt die Schreibweise nur bei Symbolen, die als external oder als public deklariert sind.

Allgemeine Optionen	Beschreibung
Größe der Symboltabelle (/kh)	Bestimmt die maximale Anzahl von Symbolen in einer Assembler-Datei (.ASM). Der zulässige Minimalwert beträgt 8.192 Byte. Das Maximum sind 32.768 Byte. Vorgabe = 8192
Maximale Symbollänge (/mv)	Legt die maximale Länge der Symbole fest, die Tasm unterscheiden kann. Die minimal zulässige Anzahl beträgt 12. Vorgabe = 12
Maximale Durchläufe (/m)	Legt die maximale Anzahl von Assemblierungsdurchläufen fest. Mit dieser Option kann erreicht werden, dass der Assembler die NOP-Anweisungen entfernt, die aufgrund von Vorwärtsreferenzen eingefügt wurden. Vorgabe = 1
Versions-ID (/u)	Setzt die Versionsemulation auf die angegebenen Versionsnummer. Vorgabe = 0
Auf unsauberem Quelltext prüfen (/p)	Überprüft auf Segmentüberschreibungen im Quelltext im geschützten Modus. Vorgabe = nicht aktiviert
.obj-Records unterdrücken (/q)	Unterdrückt für das Linken nicht erforderliche .obj-Records. Vorgabe = nicht aktiviert
Meldungen unterdrücken (/t)	Unterdrückt Meldungen, wenn erfolgreich. Vorgabe = nicht aktiviert
Quelltextzeilen in Meldungen anzeigen (/z)	Zeigt in Fehlermeldung die Quelltextzeile an.

Assembler-Direktivenoption	Beschreibung
Assembler-Direktive (/j)	Definiert eine Assembler Startup-Direktive (z.B. jIDEAL). Diese Direktive wird vor der ersten Quelltextzeile assembliert. Klicken Sie auf [...], um ein Dialogfeld zu öffnen, in dem Sie die Assembler-Direktiven verwalten können. Markieren Sie das Kontrollfeld Zusammenführen, damit die Direktiven des unmittelbaren Vorfahren übernommen werden, obwohl diese Liste nicht eigentlich verändert wird. Vorgabe = ohne Direktiven

Allgemeine Option	Beschreibung
Standard für neue Projekte	Speichert die aktuellen Einstellungen als Standardeinstellungen für neue Projekte.

2.251 Turbo Assembler Pfade und Definitionen

Projekt▶**Optionen**▶**Tasm**▶**Pfade und Definitionen**

Verwenden Sie dieses Dialogfeld dazu, Pfad- und Definitionsoptionen des Assemblers zu setzen.

Build-Konfigurationsoptionen	Beschreibung
Build-Konfiguration	Zeigt die aktive Build-Konfiguration an. Wählen Sie mit Hilfe der Drop-Down-Liste eine andere Build-Konfiguration aus.
Speichern unter...	Zeigt das Dialogfeld Speichern unter an, um die Optionen der aktuellen Konfiguration in eine Datei zu speichern, die als benannte Optionsgruppe geladen werden kann.
Laden...	Zeigt das Dialogfeld Optionsgruppe an, in dem Sie die Optionen einer benannten Optionsgruppe für die aktuelle Konfiguration übernehmen können.

Pfade und Definitionen	Beschreibung
Include-Pfad: (/i)	Gibt das Laufwerk und/oder die Verzeichnisse an, die Include-Dateien des Programms enthalten. Klicken Sie auf [...], um das Dialogfeld Dateisuchpfad einbeziehen zu öffnen, in dem Sie die Pfadliste verwalten können. Markieren Sie das Kontrollfeld Zusammenführen, damit die Pfade des unmittelbaren Vorfahren übernommen werden, obwohl diese Liste nicht eigentlich verändert wird.
Definition: (/d)	Definierte eine Liste mit definierten Symbolen in der Form Name=Wert. Klicken Sie auf [...], um das Dialogfeld Symbole definieren zu öffnen, in dem Sie die definierten Symbole verwalten können. Markieren Sie das Kontrollfeld Zusammenführen, damit die Definitionen des unmittelbaren Vorfahren übernommen werden, obwohl diese Liste nicht eigentlich verändert wird.
.obj-Ausgabeverzeichnis	Definiert das Ausgabeverzeichnis für OBJ-Dateien. Klicken Sie auf [...], um ein Dialogfeld zur Verzeichnisauswahl anzuzeigen.

Allgemeine Option	Beschreibung
Standard für neue Projekte	Speichert die aktuellen Einstellungen als Standardeinstellungen für neue Projekte.

2.252 Turbo Assembler Warnungen

[Projekt](#)▸[Optionen](#)▸[Tasm](#)▸[Warnungen](#)

Verwenden Sie dieses Dialogfeld dazu, Warnungsoptionen des Assemblers zu setzen.

Build-Konfigurationsoptionen	Beschreibung
Build-Konfiguration	Zeigt die aktive Build-Konfiguration an. Wählen Sie mit Hilfe der Drop-Down-Liste eine andere Build-Konfiguration aus.
Speichern unter...	Zeigt das Dialogfeld Speichern unter an, um die Optionen der aktuellen Konfiguration in eine Datei zu speichern, die als benannte Optionsgruppe geladen werden kann.
Laden...	Zeigt das Dialogfeld Optionsgruppe an, in dem Sie die Optionen einer benannten Optionsgruppe für die aktuelle Konfiguration übernehmen können.

Warnungsoptionen	Beschreibung
Alle einschalten (/w+)	Erzeugt alle Warnungen.
Alle ausschalten (/w-)	Deaktiviert alle Warnungen.
Ausgewählte *	Ermöglicht es, bestimmte Warnungen zu aktivieren. Dies ist die Standardeinstellung. Klicken Sie auf Alles markieren, um alle Warnungen in der Liste anzuzeigen. Klicken Sie auf Keine markieren, um alle Warnungen in der Liste zu deaktivieren. Klicken Sie auf Vorgabe, um die Standard-Warnungen in der Liste anzuzeigen.

Anmerkung: Die Standardeinstellung ist Ausgewählte.

Allgemeine Option	Beschreibung
Standard für neue Projekte	Speichert die aktuellen Einstellungen als Standardeinstellungen für neue Projekte.

2.253 Nicht verfügbare Optionen

Projekt ▶ Optionen

Einige Projektoptionen stehen in C++Builder 2007 nicht mehr zur Verfügung. Sie können eventuell über die Optionsschalter darauf zugreifen.

Im Folgenden sind diese nach Themenbereichen aufgeführt: C++-Compiler, Ressourcen-Compiler, Pascal-Compiler, IDL zu C++-Compiler, Linker, Bibliotheken und Turbo Assembler.

C++-Compiler: Unterstützung für CodeGuard-Compilierung		Beschreibung
Debug-Ebene CodeGuard	für	<p>Keine *</p> <p>CodeGuard ist aus. Dies ist die Standardeinstellung.</p> <p>Ebene 0 (-vG0)</p> <p>Aktiviert die CodeGuard-Ebene 0.</p> <p>Ebene 1 (-vG1)</p> <p>Aktiviert die CodeGuard-Ebene 1. Dadurch wird die Option -vGd aktiviert.</p> <p>Ebene 2 (-vG2)</p> <p>Aktiviert die CodeGuard-Ebene 2. Dadurch werden die Optionen -vGd und -vGt aktiviert.</p> <p>Ebene 3 (-vG3)</p> <p>Aktiviert die CodeGuard-Ebene 3. Dadurch werden die Optionen -vGd, -vGc und -vGt aktiviert.</p>

C++-Compiler: Kompatibilität		Beschreibung
Keine Beschränkungen für Verweis von Elementzeigern (-Vmv)		Wenn diese Option aktiv ist, setzt der Compiler keine Beschränkungen für den Verweis von Elementzeigern. Es wird die allgemeinste Darstellung für Elementzeiger verwendet (die allerdings nicht immer die effektivste ist).

C++-Compiler: Pfade und Definitionen		Beschreibung
Undefine aller vorherigen Namensdefinitionen (-U)		Hebt die vorherige Definition des angegebenen Bezeichners auf.

C++-Compiler: Weitere Optionen		Beschreibung
System-Header-Dateien beim Erzeugen der Abhängigkeitsinfo ignorieren (-mm)		Ignoriert bei der Erzeugung von Abhängigkeitsinformationen System-Header-Dateien.

Konsolenanwendung (-tC)	
----------------------------	--

2

C++-Compiler: Zieleinstellungen	Beschreibung
Windows-Anwendung (-tW)	Das Ziel ist eine Windows-Anwendung (entspricht -W)
Konsolenanwendung (-tWC)	Das Ziel ist eine Konsolenanwendung (entspricht -WC)
Dynamische Link-Bibliothek (-tWD)	Erzeugt eine ausführbare .DLL-Datei (entspricht -WD)
32 Bit-Multithreaded-Projekt (-tWM)	Der Compiler erstellt eine multithreaded .EXE- oder .DLL-Datei (Die Befehlszeilenoption -WM wird nur aus Gründen der Rückwärtskompatibilität unterstützt; sie hat dieselbe Funktionalität wie -tWM.) Diese Option ist nicht erforderlich, wenn Sie eine Moduldefinitionsdatei in Ihre Compiler- und Linker-Befehle einfügen, welche die Art der 32-Bit-Anwendung angibt, die Sie erstellen möchten.
Generierung einer Unicode-Anwendung (-tWU)	Erzeugt eine Unicode-Anwendung.

C++-Compiler: Assembler	Beschreibung
In .ASM (-S) compilieren, dann in .OBJ zusammenstellen (-B)	Der Compiler erzeugt zunächst aus dem C++ (oder C) Quelltext eine .ASM-Datei (entspricht der Kommandozeilenoption -S). Dann ruft der Compiler TASM32 (oder den mit der Option -E angegebenen Assembler auf), um eine .OBJ-Datei aus der .ASM-Datei zu erstellen. Die .ASM-Datei wird dann gelöscht. Das Programm kann mit der Option -B nicht compilieren, wenn Ihr C- oder C++-Quelltext statische globale Variablen deklariert, die Schlüsselwörter in der Assemblierung sind. Dies ist der Fall, weil der Compiler statischen globalen Variablen keinen Unterstrich voranstellt (wie er dies bei anderen Variablen tut), und die Assemblierungs-Schlüsselwörter Fehler erzeugen, wenn der Code zusammengestellt wird.
Zu verwendenden Assembler festlegen (-E)	Stellt Anweisungen zusammen und verwendet dabei den angegebenen Dateinamen als Assembler. Der 32-Bit-Compiler verwendet TASM32 als Standard-Assembler.
Assembler-Option festlegen, z.B. (-Tx) (-T)	Übergibt die angegebenen Optionen an den Assembler, der mit der Option -E festgelegt wurde.

C++-Compiler: Unterstützung der Stapel-Compilierung	Beschreibung
Stapel-Compilierung nach n Warnungen anhalten (Vorgabe = 255) (-g)	Warnungen: "Beenden nach" bricht die Compilierung ab, sobald eine bestimmte Anzahl von Fehlern erkannt wurde. Werte von 0 bis 255 sind möglich. Die Eingabe von 0 bewirkt, dass die Compilierung fortgesetzt wird, bis entweder das Ende der Datei oder die unter "Beenden nach" angegebene Fehlerzahl erreicht ist, je nachdem, welcher Fall zuerst eintritt.

Stop für Stapel-Compilierung nach n Fehlern angeben (Vorgabe = Keiner)(-j)	<p>Fehler: "Beenden nach" bricht die Compilierung ab, sobald eine bestimmte Anzahl von Fehlern erkannt wurde. Werte von 0 bis 255 sind möglich.</p> <p>Die Eingabe von 0 bewirkt, dass die Compilierung fortgesetzt wird bis entweder das Ende der Datei oder die unter "Beenden nach" angegebene Anzahl der Warnungen erreicht ist, je nachdem, welcher Fall zuerst eintritt.</p>
Stapel-Compilierung nach erster Datei mit Fehlern abbrechen (-jb)	<p>Bricht die Stapel-Compilierung nach der ersten Datei, die einen Fehler verursacht, ab. Beispiel:</p> <p>BCC32 -c -gb *.ccp BCC32 -c -gb file1.cpp file2.cpp</p> <p>Ohne das Flag -jb wird die Stapel-Compilierung bis zur nächsten angegebenen Datei fortgeführt, auch wenn eine vorherige Datei einen Fehler verursacht hat.</p>

C++ Compiler: Template-Optionen	Beschreibung
Definitionen für alle Template-Instanzen erzeugen und Duplikate zusammenführen (-Jgd)*	<p>Der Compiler erzeugt public (globale) Definitionen für alle Template-Instanzen. Wenn dieselbe Template-Instanz von mehreren Modulen erzeugt wird, fügt der Linker die Einzelinstanzen automatisch zu einer einzigen Kopie dieser Instanz zusammen.</p> <p>Für die Erzeugung der Instanzen benötigt der Compiler den Funktionsrumpf (bei einer Template-Funktion) oder die Rümpfe der Elementfunktionen und Definitionen für statische Datenelemente (bei einer Template-Klasse). Diese sind in der Regel in einer Header-Datei enthalten.</p> <p>Template-Instanzen lassen sich so einfach erstellen.</p>
Alle erforderlichen Instantiierungen in der C++ Syntax ausgeben (-Jgi)	Druckt alle erforderlichen Instantiierungen aus (mit C++ Syntax)
Externe Referenzen für alle Template-Instanzen erzeugen (-Jgx)	<p>Der Compiler erzeugt externe Referenzen zu allen Template-Instanzen.</p> <p>Für Template-Instanzen, die gelinkt werden müssen, muss eine explizite Instantiierungsdirektive in mindestens einem anderen Modul enthalten sein.</p>

Ressourcen-Compiler: Weitere Optionen	Beschreibung
Anweisungsdateiname (@)	Übernimmt Anweisungen von der angegebenen Befehlsdatei.
Zu compilierende Ressourcendatei (.RC)	Liste von zu compilierenden Ressourcendateien.

Ressourcen-Compiler: Ausgabeeinstellungen	Beschreibung
Ausgabedatei (.RES) (-fo)	Benennt die Ausgabedatei (.RES) um. (Standardmäßig erstellt BRCC32 die Ausgabedatei (.RES) mit demselben Namen wie die Eingabedatei (.RC).)

Pascal-Compiler: : Map-Dateioptionen		Beschreibung
Detaillierte Map-Datei (-GD)		Erstellt eine detaillierte Map-Datei.
Map-Datei mit publics (-GP)		Erstellt eine Map-Datei mit publics.
Map-Datei mit Segmenten (-GS)		Erstellt eine Map-Datei mit Segmenten.
Keine Map-Datei*		Keine Map-Datei angegeben.

Pascal-Compiler: Pfade und Definitionen		Beschreibung
Suchpfad für Objektdateien (-O)		Legt den Suchpfad für die Objektdateien fest.
Suchpfad für Ressourcendateien (-R)		Legt den Suchpfad für die Ressourcendateien fest.
Unit-Suchpfade (-U)		Legt die Unit-Suchpfade fest.

Pascal-Compiler: Weitere Optionen		Beschreibung
Alle Units erzeugen (-B)		Erzeugt alle Units.
Fehler suchen (-F)		Sucht den angegebenen Fehler.
Geänderte Units erzeugen (-M)		Erzeugt geänderte Units.
Auch nach 8.3-Namen suchen (-P)		Sucht auch nach 8.3-Namen.
Ohne Meldung compilieren (-Q)		Führt eine Compilierung ohne Meldungen aus.
Compiler-Direktiven (-\$)		Setzt für die angegebenen Direktiven Compiler-Direktiven.
Importierte Daten (-\$G+)*		Aktiviert importierte Daten.
Laufzeittypinfo (-\$M+)		Aktiviert die Laufzeittypinfo.
Aufzählungen Byte-Größe (-\$Z1)*	in	Aktiviert Aufzählungen in Byte-Größe.
Aufzählungen Word-Größe (-\$Z2)	in	Aktiviert Aufzählungen in Word-Größe.
Aufzählungen Double-Word-Größe (-\$Z4)	in	Aktiviert Aufzählungen in Double-Word-Größe.
Exportsymbole (-\$ObjExportAll On)		Exportiert Symbole.

Real-Typkompatibilität (-\$REALCOMPATIBILITY ON)	Aktiviert die Real-Typkompatibilität.
--	---------------------------------------

Pascal-Compiler: Ausgabeeinstellungen für EXE- und DLL-Dateien	Beschreibung
Konsolenziel (-CC)	Gibt das Konsolenziel aus.
GUI-Ziel (-CG)*	Gibt das GUI-Ziel aus.
Debug-Informationen in der Ausgabe (-V)	Zeigt die Debug-Informationen in der Ausgabe an.
Remote-Debug-Symbole (.rsm) erzeugen (-VR)	Erzeugt externe Debug-Symbole.

Pascal-Compiler: Linker-Arbeitsspeicher	Beschreibung
Image-Basisadresse setzen (-K)	Setzt die Image-Basisadresse für die angegebene Adresse.

Pascal-Compiler: Meldungen	Beschreibung
Ausgabe von Hinweismeldungen (-H)*	Gibt Hinweise aus.
Ausgabe von Warnungen (-W)*	Gibt Warnmeldungen aus.

IDL zu C++ Compiler	Beschreibung
Allgemein	"IDL zu C++-Compiler" ist in dem Produkt nicht mehr vorhanden.

Linker: Linken	Beschreibung
Banner unterdrücken (-q)	Unterdrückt das Banner.
Zum Linken benötigte Zeit anzeigen (-t)	Zeigt die Dauer des Linkvorgangs an.

Linker: Ausgabeeinstellungen	Beschreibung
Exe-Datei	Der Name, den Sie einer ausführbaren Datei (.EXE oder .DLL) geben möchten. Wenn Sie für die ausführbare Datei keinen Namen angeben, leitet ILINK32 den Namen der ausführbaren Datei ab, indem an den ersten aufgeführten Objektdateinamen die Endung .EXE oder .DLL angefügt wird. (Der Linker setzt die Erweiterung .EXE voraus bzw. fügt sie an, wenn keine vorhanden ist.) Ebenso fügt er an dynamische Link-Bibliotheken die Endung .DLL an, wenn diese nicht existiert.)
Map-Datei	Name, den Sie der Map-Datei geben möchten. Sofern kein Name angegeben wird, erhält die Map-Datei denselben Namen wie die EXE-Datei (aber mit der Erweiterung .MAP) (Der Linker fügt die Erweiterung .MAP an, wenn keine Erweiterung vorhanden ist).

Linker: Anwendungstyp	Beschreibung
32-Bit Windows-Anwendung (-aa)	Erzeugt eine ausführbare Datei im geschützten Modus, die unter der 32-Bit Windows API ausgeführt wird.
Windows-Gerätetreiber (-ad)	Der Anwendungstyp wird auf NATIVE gesetzt und die Image-Prüfsumme wird berechnet und gesetzt.
Konsolenanwendung (-ap)	Erzeugt eine 32-Bit ausführbare Datei im geschützten Modus, die im Konsolenmodus ausgeführt wird.

Linker: Eingabeeinstellungen	Beschreibung
Objektdateien	Die OBJ-Dateien, die Sie linken möchten. Geben Sie den Pfad an, wenn die Dateien sich nicht im aktuellen Verzeichnis befinden. (Der Linker fügt die Erweiterung .OBJ an, wenn keine Erweiterung vorhanden ist).
Bibliotheksdateien	Die Bibliotheksdateien, die zur Link-Zeit eingefügt werden sollen. Verwenden Sie keine Kommas, um die aufgeführten Dateien zu trennen. Wenn sich eine Datei nicht im aktuellen Verzeichnis oder dem Suchpfad befindet, dann müssen Sie diese Datei in die Link-Anweisung einfügen. (Der Linker fügt die Erweiterung .LIB an, wenn keine Erweiterung vorhanden ist). Die Reihenfolge, in der die Bibliotheken aufgeführt werden, ist sehr wichtig; stellen Sie sicher, dass die Reihenfolge eingehalten wird, die in dieser Liste angegeben ist: <ol style="list-style-type: none"> 1. CodeGuard-Bibliotheken (falls erforderlich) 2. Listen Sie alle Ihre benutzerdefinierten Bibliotheken auf und beachten Sie dabei: wenn eine Funktion mehr als einmal definiert ist, verwendet der Linker die erste Definition, auf die er trifft. 3. IMPORT32.LIB (wenn Sie eine ausführbare Datei erstellen, die die Windows API verwendet) 4. Mathematische Bibliotheken 5. Laufzeitbibliotheken
Ressourcendateien	Eine Liste von RES-Dateien (compilierte Ressourcendateien), die mit der ausführbaren Datei verbunden werden. (Der Linker fügt die Erweiterung .RES an, wenn keine Erweiterung vorhanden ist).
Def-Datei	Die Datei zur Moduldefinition für eine ausführbare Windows-Datei. Wenn Sie keine Moduldefinitionsdatei (DEF) angeben und die Option /Twe oder /Twd verwendet haben, erstellt der Linker eine Anwendung, die auf den Standardeinstellungen basiert. (Der Linker fügt die Erweiterung .DEF an, wenn keine Erweiterung vorhanden ist).

Linker: Weitere Optionen	Beschreibung
Image-Kommentar-String festlegen (-GC)	Fügt Kommentar-Strings in das Image ein. Diese Strings werden in das Image direkt nach der Objekttabelle in den PE-Datei-Header eingefügt. Sie können mehrere Strings festlegen.

Linker: Packages	Beschreibung
Basisname des Package (-GB)	Weist dem Package einen Basisnamen zu.
Statisches Package (-GI)	Erzeugt ein statisches Package.
Nur-Entwurfszeit-Package (-Gpd)	Erzeugt ein nur-Entwurfszeit-Package. (Wenn weder /Gpr noch /Gpd verwendet werden, funktioniert das Package sowohl zur Entwurfs- als auch zur Laufzeit).
Nur-Laufzeit-Package (-Gpr)	Erzeugt ein Laufzeit-Package. (Wenn weder /Gpr noch /Gpd verwendet werden, funktioniert das Package sowohl zur Entwurfs- als auch zur Laufzeit).

Linker: Pfade und Definitionen	Beschreibung
Bibliothekssuchpfad (-L)	<p>Legt die Verzeichnisse fest, die der Linker durchsucht, wenn für ein LIB-Modul in der Compilier/Link-Anweisung kein expliziter Pfad angegeben ist.</p> <p>Um den Bibliothekssuchpfad anzugeben, ist folgende Befehlszeilensyntax erforderlich: <code>/L<Pfadangabe>[;<Pfadangabe>][...]</code></p> <p>Der Linker verwendet die angegebenen Bibliothekssuchpfade, wenn kein expliziter Pfad für die LIB-Datei angegeben ist, und der Linker die Bibliothekssdatei nicht im aktuellen Verzeichnis finden kann. Der Befehl</p> <pre>ILINK32 /Lc:\mylibs;.\libs splash.\common\logo,,,utils logolib</pre> <p>beauftragt den Linker beispielsweise, zuerst das aktuelle Verzeichnis nach SPLASH.LIB zu durchsuchen. Wenn diese Datei im aktuellen Verzeichnis nicht gefunden wird, sucht der Linker im Verzeichnis C:\MYLIBS und dann im Verzeichnis .LIBs. Beachten Sie jedoch, dass der Linker die Bibliothekssuchpfade nicht verwendet, um die Datei LOGO.LIB zu suchen, da dieser Datei ein expliziter Pfad zugewiesen wurde.</p>
Objektsuchpfade festlegen (-j)	<p>Legt die Verzeichnisse fest, die der Linker durchsucht, wenn für ein Objektmodul in der Compilier/Link-Anweisung kein expliziter Pfad angegeben ist.</p> <p>Um den Objektsuchpfad anzugeben, ist folgende Befehlszeilensyntax erforderlich: <code>\j<Pfadangabe>[;<Pfadangabe>][...]</code></p> <p>Der Linker verwendet die angegebenen Objektsuchpfade, wenn kein expliziter Pfad für die Objektdatei angegeben ist, und der Linker die Objektdatei nicht im aktuellen Verzeichnis finden kann. Der Befehl</p> <pre>ILINK32 /jc:\myobjs;.\objs splash.\common\logo,,,utils logolib</pre> <p>beauftragt den Linker beispielsweise, zuerst das aktuelle Verzeichnis nach SPLASH.OBJ zu durchsuchen. Wenn diese Datei im aktuellen Verzeichnis nicht gefunden wird, sucht der Linker im Verzeichnis C:\MYOJJS und dann im Verzeichnis .OJJS. Beachten Sie jedoch, dass der Linker die Bibliothekssuchpfade nicht verwendet, um die Datei LOGO.OBJ zu suchen, da dieser Datei ein expliziter Pfad zugewiesen wurde.</p>

Linker: PE-Datei	Beschreibung
Image-Basisadresse festlegen (Adressverschiebungstabelle beibehalten) (-b)	Gibt eine Image-Basisadresse für die EXE- oder DLL-Datei an. Die Ladeadresse für das erste Objekt in der Anwendung oder Bibliothek wird, wenn möglich, auf die angegebene Nummer gesetzt, und alle nachfolgenden Objekte werden an den linearen Adressgrenzen bei 64 K ausgerichtet; interne Fixups werden nicht erkannt. Wenn die Module jedoch nicht an der angegebenen Adresse geladen werden können, wandelt das Betriebssystem diese in die Standardeinstellungen um und weist interne Fixups zu.

Linker: Windows-Anwendungstyp	Beschreibung
Windows Dynamische Link-Bibliothek (-Tpd)	Der Linker generiert eine 32-Bit Windows DLL-Datei im geschützten Modus.
Windows-Anwendung (-Tpe)	Der Linker generiert eine 32-Bit Windows EXE-Datei im geschützten Modus.
C++Builder-Package (-Tpp)	Der Linker generiert ein Package. Diese Option wird automatisch in Package-Make-Dateien eingefügt.

Bibliotheken: Weitere Optionen	Beschreibung
Importe erzwingen (-f)	Erzwingt Importe per Namen.
WEP ignorieren (-i)	Ignoriert WEP.
Modulerweiterungen entfernen (-o)	Entfernt Modulerweiterungen.
Keine Warnungen (-w)	Keine Warnungen.

Bibliotheken: Eingabeeinstellungen	Beschreibung
Quelltextdateinamen	Weist der Quelltextdatei einen Namen zu.

Bibliotheken: Ausgabeeinstellungen	Beschreibung
Bibliotheksname	Weist der Bibliothek einen Namen zu.

Turbo Assembler: Listing-Datei	Beschreibung
Kreuzreferenz in Listing-Datei erzeugen (/c)	Erzeugt eine Kreuzreferenz in Listing-Dateien. Tasm fügt die Kreuzreferenz-Informationen zur Symboltabelle am Ende der Listing-Datei hinzu.
Erweitertes Listing erzeugen (/la)	Erzeugt ein normales Listing, das zusätzlich den C/C++ Code enthält, der Basis für die .ASM-Datei ist.

Symboltabellen Listing-Datei unterdrücken (/n)	in	Unterdrückt die Symboltabellen in der Listing-Datei.
false-Bedingungen Listing-Datei aufnehmen (/x)	in	Fügt false-Bedingungen in ein Listing ein. Wenn eine Bedingung (z.B. #if, #ifndef, #ifdef) zu False ausgewertet wird, bewirkt diese Option, dass die Anweisungen innerhalb des Bedingungsblocks in der Listing-Datei erscheinen.

Turbo Assembler: Ausgabeeinstellungen		Beschreibung
Name	der Ausgabeobjektdatei	Weist der Ausgabeobjektdatei einen Dateinamen zu.
Dateiname	der Listing-Datei	Weist der Listing-Datei einen Dateinamen zu.
Dateiname	für Kreuzreferenzdatei	Weist der Kreuzreferenzdatei einen Dateinamen zu.

2.254 Information

Projekt ▶ Informationen

Verwenden Sie dieses Dialogfeld zum Überprüfen von Compiler-Informationen und des Compilierungs-Status.

Element	Beschreibung
Compilierter Quelltext	Zeigt die Gesamtanzahl der compilierten Zeilen an.
Codegröße	Zeigt die Gesamtgröße der ausführbaren Datei oder der Assemblierung ohne Fehlerinformationen an.
Datengröße	Zeigt den für die Ablage der globalen Variablen benötigten Speicher an.
Anfangs-Stackgröße	Zeigt den für die Ablage der lokalen Variablen benötigten Speicher an.
Dateigröße	Zeigt die Größe der endgültigen Ausgabedatei an.
Status	Zeigt an, ob die letzte Compilierung erfolgreich war oder fehlgeschlagen ist.

2.255 Aus dem Projekt entfernen

Projekt ▶ Aus dem Projekt entfernen

Verwenden Sie dieses Dialogfeld, um eine oder mehrere Dateien aus dem aktuellen Projekt zu entfernen.

Element	Beschreibung
Dateiliste	Markieren Sie die Datei, die entfernt werden soll. Wenn Sie mehrere Dateien markieren möchten, halten Sie bei der Auswahl die Taste STRG gedrückt.

Anmerkung: Wenn Sie eine Datei entfernen, die während der aktuellen Sitzung bearbeitet wurde, werden Sie zum Speichern der Änderungen aufgefordert. Für Dateien, an denen keine Änderungen vorgenommen wurden, wird diese Aufforderung nicht angezeigt.

Warnung: Entfernen Sie die Datei aus dem Projekt, bevor Sie sie von der Festplatte löschen. Nur dann kann die erforderliche Aktualisierung der Projektdatei vorgenommen werden.

2.256 Optionen

Projektverwaltung ▶ Eine Satelliten-Assemblierung mit der rechten Maustaste anklicken ▶ Befehl Optionen

Verwenden Sie dieses Dialogfeld, um Optionen des Assemblierungs-Linkers (`al.exe`) für die in der Projektverwaltung ausgewählte Satelliten-Assemblierung zu setzen.

Element	Beschreibung
Kultur	Legt den Kultur-String fest, der der Assemblierung zugeordnet werden soll. Wählen Sie eine Kultur aus der Dropdown-Liste. Enspricht der Option <code>/culture</code> .
Firma	Legt einen String für das Feld Firma in der Assemblierung fest. Enspricht der Option <code>/company</code> .
Konfiguration	Legt einen String für das Feld Konfiguration in der Assemblierung fest. Enspricht der Option <code>/configuration</code> .
Copyright	Legt einen String für das Feld Copyright in der Assemblierung fest.
Beschreibung	Legt einen String für das Feld Beschreibung in der Assemblierung fest. Enspricht der Option <code>/description</code> .
Warenzeichen	Legt einen String für das Feld Trademark in der Assemblierung fest. Enspricht der Option <code>/trademark</code> .
Produkt	Legt einen String für das Feld Produkt in der Assemblierung fest. Enspricht der Option <code>/product</code> .
Produktversion	Legt einen String für das Feld Produktversion in der Assemblierung fest. Enspricht der Option <code>/productversion</code> .
Titel	Legt einen String für das Feld Titel in der Assemblierung fest. Enspricht der Option <code>/title</code> .
Dateiversion	Legt einen String für das Feld Dateiversion in der Assemblierung fest. Enspricht der Option <code>/fileversion</code> .
Nachweis	Legt die Datei fest, die in der Assemblierung mit dem Ressourcennamen <code>Security.Nachweis</code> eingebettet werden soll. Enspricht der Option <code>/evidence</code> .
Schlüsseldatei	Legt die Datei fest, die ein Schlüsselpaar oder einen als public deklarierten Schlüssel enthält, mit dem die Assemblierung signiert werden soll. Der Compiler fügt den als public deklarierten Schlüssel in das Assemblierungsmanifest ein und signiert dann die endgültige Assemblierung mit dem privaten Schlüssel. Enspricht der Option <code>/keyfile</code> .
Schlüsselname	Legt einen Container für ein Schlüsselpaar fest. Dadurch wird die Assemblierung mit einem starken Namen signiert, indem ein als public deklarerter Schlüssel in das Assemblierungsmanifest eingefügt wird. Enspricht der Option <code>/keyname</code> .

Tip: Die im Feld Optionen aufgeführten Attribute können in Reflection angezeigt werden.

Weitere Informationen zum Assemblierungs-Linkern, Satelliten-Assemblierungen und Reflection finden Sie in der Online-Hilfe

zum .NET Framework SDK.

2.257 Symbol wählen

Verwenden Sie dieses Dialogfeld zum Auswählen eines Bitmaps, das Ihre Template im Dialogfeld Objektgalerie repräsentieren soll.

Sie können Bitmaps beliebiger Größe verwenden, die jedoch immer im Format 60 x 40 Pixel angezeigt werden.

2.258 Optionen für Distribution über das Web

Projekt ▶ Optionen für Web-Distribution

Verwenden Sie dieses Dialogfeld, um ein fertiges ActiveX-Steuerelement oder ein ActiveForm für die Distribution an den Windows Web-Server zu übertragen.

Tip: Legen Sie diese Optionen fest, bevor Sie das ActiveX-Projekt compilieren und mit dem Befehl [Projekt ▶ Distribution über das Web](#) weitergeben.

Registerkarte Projekt

Verwenden Sie diese Seite, um Verzeichnispfade, eine URL, die CAB-Dateikompression und Versionsinformationen festzulegen.

Element	Beschreibung
Zielverzeichnis	Gibt den Speicherort der ActiveX-Bibliotheksdatei als Pfadangabe im Web-Server fest. Es kann sich dabei um einen Standardpfadnamen oder eine UNC-Pfadangabe handeln. Klicken Sie die Schaltfläche Durchsuchen an, um das gewünschte Verzeichnis, z.B. C:\INETPUB\wwwroot , zu suchen.
Ziel-URL	Gibt die URL der ActiveX-Bibliotheksdatei an. In der Dokumentation Ihres Web-Servers ist beschrieben, wie URLs, z.B. http://mymachine.borland.com/ , festgelegt werden.
HTML-Verzeichnis	Gibt den Speicherort an, wo die HTML-Datei, die eine Referenz auf das ActiveX-Steuerelement enthält, erzeugt werden soll. Es kann sich dabei um einen Standardpfadnamen oder eine UNC-Pfadangabe handeln. Klicken Sie die Schaltfläche Durchsuchen an, um das gewünschte Verzeichnis, z.B. C:\INETPUB\wwwroot , zu suchen.
CAB-Dateikompression verwenden	Komprimiert die ActiveX-Bibliothek und alle erforderlichen Packages und zusätzlichen Dateien. Die CAB-Dateikompression speichert Dateien in einer Dateibibliothek, die die Download-Dauer um bis zu 70 Prozent verringern kann.
Versionsinfo der Datei übernehmen	Bezieht die Versionsinformationen der Seite Versionsinfo des Dialogfeldes Projektoptionen ein.
Versionsnummer auto-inkrementieren	Erhöht die Versionsnummer bei jedem Aufruf des Befehls Projekt ▶ Distribution über das Web automatisch um eins. Dadurch wird auch der Wert auf der Seite Versionsinfo des Dialogfeldes Projektoptionen aktualisiert.
Benötigte Dateien	Gibt alle auf der Seite Packages aufgeführten Packages zusammen mit dem Projekt weiter.
Zusätzliche Dateien	Gibt alle auf der Seite Zusätzliche Dateien aufgeführten Dateien zusammen mit dem Projekt weiter.

Registerkarte Packages

Verwenden Sie diese Seite, um festzulegen, welche Packages mit Ihrem Projekt weitergegeben werden müssen und wie sie für das Deployment bereitgestellt werden. Für jedes Package können eigene Optionen festgelegt werden, die die Vorgaben auf der Seite Projekt überschreiben. Packages, die mit diesem Programm ausgeliefert werden, sind mit der Code-Unterzeichnung Borland versehen.

Anmerkung: Sie müssen auf der Seite Projekt das Kontrollkästchen Benötigte Dateien weitergeben markieren, um diese Dateien einzubeziehen. Ansonsten werden diese Packages nicht weitergegeben und Sie können in der Packages-Liste keine Packages auswählen.

Element	Beschreibung
Verwendete Packages	Führt die Packages auf, die für Ihr ActiveX-Bibliotheksprojekt erforderlich sind. Markieren Sie ein Package in dieser Liste, um dessen Optionen zu ändern.
In separatem CAB komprimieren	Erzeugt eine separate .cab-Datei für das Package.
In Projekt-CAB komprimieren	Bezieht das Package in die .cab-Projektdatei ein.
Datei-Versionsinfo verwenden	Wenn das Package eine Versionsinfo-Ressource einbezieht, wird die Versionsinformation in dieser Ressource zu der .inf-Datei des Projekts hinzugefügt.
Ziel-URL	Legt die URL für die Package-Datei fest. Wenn das Feld leer ist, nimmt der Web-Browser an, dass die Datei bereits auf dem Client-Rechner vorhanden ist. Wenn der Client das Package nicht enthält, kann der Download der ActiveX-Bibliothek nicht ausgeführt werden.
Zielverzeichnis	Legt das Verzeichnis auf dem Server fest, in das das Package geschrieben werden soll. Es kann sich dabei um einen Standardpfadnamen oder eine UNC-Pfadangabe handeln. Ein leeres Feld gibt an, dass die Datei bereits vorhanden ist und nicht überschrieben werden sollte.

Registerkarte Zusätzliche Dateien

Verwenden Sie diese Seite, um festzulegen, welche Dateien außer Packages mit Ihrem Projekt weitergegeben werden müssen und wie sie für das Deployment bereitgestellt werden. Auf dieser Seite können Sie weitere Dateien hinzufügen und Optionen für einzelne Dateien festlegen. Diese Optionen überschreiben die Vorgaben der Seite Projekt.

Anmerkung: Sie müssen auf der Seite Projekt das Kontrollkästchen Benötigte Dateien weitergeben markieren, um diese Dateien einzubeziehen. Ansonsten werden diese Dateien nicht weitergegeben und Sie können in der Dateiliste keine Dateien auswählen.

Element	Beschreibung
Verbundene Dateien	Führt die Dateien (keine Packages) auf, die für Ihr ActiveX-Bibliotheksprojekt erforderlich sind. Dateien werden durch Anklicken der Schaltfläche Hinzufügen hinzugefügt. Eine markierte Datei lässt sich entfernen, indem Sie die Schaltfläche Entfernen anklicken. Markieren Sie eine Datei in dieser Liste, um deren Optionen zu ändern.
In separatem CAB komprimieren	Erzeugt eine separate .cab-Datei für das Package.
In Projekt-CAB komprimieren	Bezieht das Package in die .cab-Projektdatei ein.
Datei-Versionsinfo verwenden	Wenn das Package eine Versionsinfo-Ressource einbezieht, wird die Versionsinformation in dieser Ressource zu der .inf-Datei des Projekts hinzugefügt.
Ziel-URL	Legt die URL für die Package-Datei fest. Wenn das Feld leer ist, nimmt der Web-Browser an, dass die Datei bereits auf dem Client-Rechner vorhanden ist. Wenn der Client das Package nicht enthält, kann der Download der ActiveX-Bibliothek nicht ausgeführt werden.
Zielverzeichnis	Legt das Verzeichnis auf dem Server fest, in das das Package geschrieben werden soll. Es kann sich dabei um einen Standardpfadnamen oder eine UNC-Pfadangabe handeln. Ein leeres Feld gibt an, dass die Datei bereits vorhanden ist und nicht überschrieben werden sollte.

2.259 Durchsuchen (Dialogfeld)

Das Dialogfeld **Durchsuchen** hat in Abhängigkeit davon, wo Sie es öffnen, unterschiedliche Verwendungszwecke.

Sie können es für eine der folgenden Aufgaben verwenden:

- Zum Laden einer vorhandenen Datei in den OLE-Container. Die von Ihnen gewählte Datei muss mit einer Anwendung verknüpft sein, die als OLE-Server verwendbar ist.
- Zum Auswählen eines Symbols, das ein OLE-Objekt auf dem Formular repräsentiert.

Durchsuchen: Um das Dialogfeld **Durchsuchen** zu öffnen, gehen Sie nach einem der folgenden Punkte vor:

- Klicken Sie im Dialogfeld **Objekt einfügen** auf **Durchsuchen**, wenn Sie **Aus Datei erstellen** ausgewählt haben.
- Klicken Sie im Dialogfeld **Anderes Symbol** auf **Durchsuchen**.

Optionen im Dialogfeld Durchsuchen

Dateiname

Geben Sie den Namen der zu ladenden Datei oder die im Listenfeld Dateien als Filter zu verwendenden Platzhalterzeichen ein.

Dateien

Zeigt die Dateien des aktuellen Verzeichnisses an, die mit den Platzhalterzeichen im Eingabefeld **Dateiname** oder mit dem Dateityp in dem Kombinationsfeld **Dateityp** übereinstimmen.

Dateityp

Wählen Sie den Dateityp, den Sie als OLE-Server verwenden möchten. Standardmäßig werden im Listenfeld **Dateien** alle Dateien des aktuellen Verzeichnisses angezeigt.

Speichern in

Wählen Sie das Verzeichnis aus, dessen Inhalt Sie zu sehen wünschen. Im Listenfeld **Dateien** erscheinen die Dateien des aktuellen Verzeichnisses, die zu den Platzhalterzeichen im Eingabefeld **Dateiname** oder zum Dateityp aus dem Kombinationsfeld **Dateityp** passen.

Laufwerke

Wählen Sie das aktuelle Laufwerk. Die Verzeichnisstruktur des aktuellen Laufwerks erscheint im Listenfeld Speichern in.

2.260 Anderes Symbol

Verwenden Sie dieses Dialogfeld zur Angabe eines Symbols und einer Beschriftung für das Objekt, das Sie auf dem Formular platzieren.

So öffnen Sie das Dialogfeld Anderes Symbol:

1. Aktivieren Sie Als Symbol im Dialogfeld Objekt einfügen.
2. Klicken Sie auf Anderes Symbol.

Element	Beschreibung
Symbol	<p>Wählen Sie das Symbol aus, das verwendet werden soll: Aktuell Verwendet das aktuelle Symbol Standard Verwendet das Vorgabesymbol. Aus Datei Ermöglicht Ihnen die Angabe eines Symbols unter Verwendung eines vollständigen Pfadnamens. Wenn Sie den Namen oder Pfad des Symbols nicht kennen, klicken Sie auf Durchsuchen, um das Dialogfeld Durchsuchen zu öffnen. Das Anzeigefeld unterhalb des Eingabefeldes zeigt alle verfügbaren Symbole in der angegebenen Datei. Wählen Sie eines dieser Symbole aus.</p>
Label	Geben Sie die Beschriftung ein, die unter dem Symbol auf dem Formular erscheinen soll.

Durchsuchen	Klicken Sie auf Durchsuchen, um das Dialogfeld Durchsuchen zu öffnen, in dem Sie ein Symbol auswählen können, das das eingefügte Objekt auf dem Formular repräsentieren soll.
Anzeige eines Beispieldsymbols	Zeigt das Symbol und die Beschriftung so an, wie sie auf dem Formular erscheinen werden.

Siehe auch

Symbol anzeigen (siehe Seite 603)

2.261 Farbeditor

Verwenden Sie den Farbeditor zur Spezifizierung oder Definition einer Farbe für die markierte Komponente. Änderungen, die Sie mithilfe des Farbeditors vornehmen, spiegeln sich in der Eigenschaft Color einer Komponente wider.

So öffnen Sie den Farbeditor:

1. Wählen Sie eine Komponente oder das Formular aus.
2. Doppelklicken Sie in der Wertespalte der Eigenschaft Color oder einer anderen Eigenschaft, die den Farbeditor verwendet.

Grundfarben

Zeigt eine Auswahl der Grundfarben an. Klicken Sie auf eine Farbe, um diese der markierten Komponente zuzuweisen.

Benutzerdefinierte Farben

Zeigt die von Ihnen erstellten Farben an. Sie können benutzerdefinierte Farben erzeugen, indem Sie auf **Farben definieren** klicken.

Farben definieren

Klicken Sie auf **Farben definieren**, um den Farbeditor um die Optionen zu erweitern, die Ihnen die Erzeugung eigener Farben ermöglichen.

Farbfeld

Zeigt das Spektrum verfügbarer Farben an. Die Fadenkreuze bezeichnen die aktuelle Farbe.

Klicken Sie an einer beliebigen Stelle im Farbfeld oder ziehen Sie die Kreuze, um eine Farbe auszuwählen. Wenn Sie hier eine Farbe auswählen und dann auf **Farbe hinzufügen** klicken, wird die ausgewählte Farbe einem der Felder für benutzerdefinierte Farben hinzugefügt, so dass Sie sie erneut verwenden können.

Farbe / Basis

Zeigt die aktuell ausgewählte Farbe zusammen mit der nächstliegenden Grundfarbe an. Doppelklicken Sie auf die Grundfarbe, um diese als aktuelle Farbe zu wählen.

Farbton

Geben Sie einen Wert für den Farbton ein. Der Farbton ist die "tatsächliche" Farbe, also beispielsweise Rot, Gelbgrün oder Purpur. Der Farbton bezieht sich auf die Farbe ohne Berücksichtigung der Sättigung oder der Helligkeit (Leuchtkraft).

Sättigung (Sätt)

Geben Sie einen Wert für die Sättigung ein. Die Sättigung bezieht sich darauf, wieviel Grau in der Farbe enthalten ist. Das Feld **Sätt** zeigt die Sättigung zwischen 0 (mittelgrau) und 240 (reine Farbe).

Anmerkung: Die Sättigung hat Einfluß darauf, wie klar eine Farbe ist. Die Leuchtkraft beeinflußt die Helligkeit der Farbe.

Leuchtkraft (Hell) und der entsprechende Schieberegler

Geben Sie einen Wert für die Leuchtkraft ein, oder ziehen Sie den Zeiger auf dem Gleitregler, um die Leuchtkraft festzulegen. Die Leuchtkraft ist die Helligkeit einer Farbe. Das Feld **Hell** zeigt die Leuchtkraft zwischen 0 (schwarz) und 240 (weiß). Die Spalte rechts neben dem Farbfeld zeigt den Bereich der Leuchtkraft für die aktuelle Farbe. Verschieben Sie den Pfeil rechts neben der Spalte nach oben oder nach unten, um die Leuchtkraft einzustellen. Beim Ändern der Leuchtkraft ändern sich auch die Farbwerte für Rot/Grün/Blau.

2 Rot/Gün/Blau

Geben Sie Werte für den Anteil von Rot, Grün und Blau ein, den Sie in Ihrer Farbe haben möchten. Die Werte dieser Felder zeigen das Verhältnis von Rot, Grün und Blau in der aktuellen Farbe. Dies wird auch als RGB-Farbe bezeichnet. Die Werte einer RGB-Farbe müssen zwischen 0 und 255 liegen.

Farben hinzufügen

Klicken Sie auf diese Schaltfläche, um die von Ihnen definierte Farbe der Palette für benutzerdefinierte Farben des Farbeditors hinzuzufügen.

2.262 DDE-Info (Dialogfeld)

Verwenden Sie das Dialogfeld **DDE-Info**, um zur Entwurfszeit eine DDE-Server-Anwendung und ein Topic für eine DDE-Konversation festzulegen.

So öffnen Sie das Dialogfeld DDE-Info:

1. Platzieren Sie eine DDECClientConv-Komponente im Formular.
2. Markieren Sie diese Komponente, und
 - klicken Sie entweder auf die Ellipsen-Schaltfläche in der Wertespalte der Eigenschaft DdeService oder der Eigenschaft DdeTopic, oder
 - doppelklicken Sie in der Wertespalte der Eigenschaft DdeService oder DdeTopic.

Optionen im Dialogfeld DDE-Info

Folgende Optionen stehen im Dialogfeld **DDE-Info** zur Verfügung:

DdeService

Geben Sie die Server-Anwendung für die DDE-Konversation an. Die von Ihnen angegebene Anwendung wird in die Wertespalte der Eigenschaft DdeService eingetragen.

Sie brauchen für die Server-Anwendung keine Dateinamenserweiterung anzugeben.

Befindet sich das Verzeichnis, das Ihre Anwendung enthält, nicht in Ihrem Pfad, so müssen Sie den vollständigen Pfad angeben.

DdeTopic

Geben Sie hier das Thema für eine DDE-Konversation ein. Dies ist eine für den Server identifizierbare Dateneinheit, die den verknüpften Text enthält. Beispielsweise könnte das Thema der Dateiname eines Arbeitsblattes sein.

Ist der Server eine VCL-Anwendung, so bildet der Name des Formulars, das die zu verknüpfenden Daten beinhaltet, das Thema.

Befindet sich das Verzeichnis, das das Thema enthält, nicht in Ihrem Pfad, so müssen Sie den vollständigen Pfad angeben.

Verknüpfung einfügen

Klicken Sie auf **Verknüpfung einfügen**, um den Namen der Anwendung und den Dateinamen vom Inhalt der Zwischenablage in die Eingabefelder für die Anwendung bzw. die Datei einzufügen.

Diese Schaltfläche ist nur aktiv, wenn die Zwischenablage Daten einer Anwendung enthält, die als DDE-Server dienen kann.

2.263 Vorlagen löschen

Kontextmenü Menü-Designer ▶ Vorlagen löschen

Verwenden Sie dieses Dialogfeld dazu, vordefinierte Templates aus der Liste im Dialogfeld Template einfügen zu entfernen.

Menü Bearbeiten	Entfernt das Standard Bearbeitungsmenü mit den Elementen Rückgängig, Wiederholen, Ausschneiden, Kopieren, Einfügen, Inhalt einfügen, Suchen, Ersetzen, Gehe zu, Beziehungen und Objekte.
Menü Datei	Entfernt das Standard-Dateimenu mit den Elementen Neu, Öffnen, Speichern, Speichern unter, Drucken, Drucker einrichten und Beenden.
Menü Datei (für TextEdit-Beispiel)	Entfernt das Dateimenu mit den Standardelementen und schließt auch den Menüpunkt Schließen ein.
Menü Hilfe	Entfernt das Hilfemenü mit den Elementen Inhalt, Themenbezogene Hilfe , Hilfe verwenden und Info.
Menü Hilfe (erweitert)	Entfernt das Hilfemenü mit den Standardelementen plus weiteren Elementen wie Index, Anweisungen, Anleitungen, Tastatur und Tutorial.
MDI-Framemenü	Removes the set of menus that contain the same items as the File, Edit, Window, and Help (Expanded) menus.
Fenster	Entfernt das Menü Fenster mit den Elementen Neues Fenster, Nebeneinander, Überlappend, Alle anordnen, Schließen und Anzeigen.

2.264 Filter-Editor

Verwenden Sie den Filter-Editor zur Definition von Filtern für die Komponente OpenDialog und die Komponente SaveDialog. Diese allgemeinen Dialogfelder verwenden den Wert von **Filter** im Kombinationsfeld **Dateityp** zur Anzeige bestimmter Dateien in einem Listenfeld **Dateien**.

Mit dem Filter-Editor bearbeiten Sie die Eigenschaft Filter.

So öffnen Sie den Filter-Editor:

1. Platzieren Sie eine OpenDialog- oder eine SaveDialog-Komponente im Formular.
2. Markieren Sie die Komponente und führen Sie eine der folgenden Aktionen durch:
 - Klicken Sie entweder auf die Ellipsen-Schaltfläche in der Wertespalte der Eigenschaft Filter, oder
 - doppelklicken Sie in der Wertespalte der Eigenschaft Filter.

Filtername

Geben Sie in diese Spalte den Namen des Filters ein, der im Kombinationsfeld **Dateityp** erscheinen soll.

Filter

Geben Sie in diese Spalte Platzhalterzeichen und Namenserweiterungen zur Definition Ihres Filters ein. So zeigt *.txt beispielsweise nur Dateien mit der Namenserweiterung .txt an.

Um mehrere Namenserweiterungen für Ihren Filter anzugeben, trennen Sie diese jeweils durch ein Semikolon (;).

2.265 Schriftarten-Editor

Verwenden Sie den Schriftarten-Editor, um zur Entwurfszeit eine Schriftart bzw. Schriftartattribute für die markierte Komponente oder das ausgewählte Formular festzulegen. Änderungen, die Sie mithilfe des Schriftarten-Editors durchführen, spiegeln sich in der Eigenschaft Font einer Komponente wider.

So öffnen Sie den Schriftarten-Editor:

Gehen Sie zum Öffnen des Schriftarten-Editors folgendermaßen vor:

1. Wählen Sie eine Komponente oder das Formular aus.

2. Verwenden Sie eines der folgenden Verfahren:

- Klicken Sie auf die Ellipsen-Schaltfläche in der Wertespalte der Eigenschaft Font oder einer anderen Eigenschaft, die den Schriftarten-Editor verwendet.
- Doppelklicken Sie in der Wertespalte der Eigenschaft Font oder einer anderen Eigenschaft, die den Schriftarten-Editor verwendet.

Schrift

Wählen Sie aus der Liste verfügbarer Schriftarten eine Schriftart aus, die Sie in Ihrer Anwendung benutzen möchten.

Schriftschnitt

Wählen Sie einen Schriftschnitt für die Schriftart. In diesem Kombinationsfeld werden nur die Schriftschnitte angezeigt, die für die gewählte Schriftart verfügbar sind. Für die meisten der verfügbaren Schriftarten gibt es vier mögliche Schriftschnitte:

- Standard
- Kursiv
- **Fett**
- **Fett-kursiv**

Schriftgrad

Wählen Sie eine Größe für die Schriftart (in Punkten). In diesem Kombinationsfeld werden nur die Schriftgrößen angezeigt, die für die gewählte Schriftart verfügbar sind.

Effekte

Markieren Sie diese Optionen, um den Text durchgestrichen oder unterstrichen anzuzeigen.

Farbe

Wählen Sie eine Farbe für die Schriftart. In diesem Kombinationsfeld werden alle für die ausgewählte Schriftart verfügbaren Farben angezeigt.

Beispiel

Zeigt ein Muster der ausgewählten Schriftart an, bevor Sie diese auf die Komponenten anwenden. Die Schriftart in diesem Bereich wird immer dann aktualisiert, wenn Sie eine Änderung an den Schriftarteneinstellungen vornehmen.

Skript

Zeigt die verfügbaren Sprachskripts für die ausgewählte Schriftart an.

2.266 Aktionsmanager-Editor

Verwenden Sie den Aktionsmanager-Editor, um zur Entwurfszeit *ActionBands*-Menüs und -Symbolleisten über eine *TActionManager*-Komponente Aktionen hinzufügen.

Zum Anzeigen des Aktionsmanager-Editors markieren Sie das *TActionManager*-Objekt und doppelklicken oder klicken mit der rechten Maustaste und wählen Anpassen.

Registerseite Symbolleisten

Mit der Registerkarte Symbolleisten können Sie schnell Symbolleisten (des Typs *TActionToolBar*) zu Ihrer Anwendung hinzufügen. Klicken Sie dazu nur die Schaltfläche Neu an. Mit der Schaltfläche Löschen entfernen Sie eine unerwünschte Symbolleiste wieder. Alle *ActionBands*-Symbolleisten der Anwendung werden in dem Feld Symbolleisten angezeigt. Zum Ein- oder Ausblenden markieren Sie die jeweilige Leiste bzw. entfernen Sie die Markierung wieder. Die Beschriftungsoptionen für die Symbolleisten ändern Sie, indem Sie sie im Feld Symbolleistenoptionen bearbeiten.

Registerseite Aktionen

Am oberen Rand des Editors befindet sich eine Symbolleiste mit vier Schaltflächen. Dies sind:

Schaltfläche Beschreibung

Neue Aktion Einfügen einer neuen Aktion in die Liste. Durch Klicken auf den Dropdown-Pfeil neben der Schaltfläche können Sie wählen, ob eine neue selbstdefinierte oder eine vordefinierte Standardaktion hinzugefügt wird. Wenn Sie Standardaktion wählen, wird ein Dialogfeld mit einer Liste der Standardaktionen geöffnet.

Löschen Löschen der aktuell ausgewählten Aktion.

Pfeilschaltflächen Ändern der Position der aktuell ausgewählten Aktion in der Liste.

Der untere Bereich der Registerseite Aktionen enthält zwei Listenfelder, die die aktuelle Liste der Aktionen enthalten. Die erste Liste gibt den Wert der Eigenschaft *Category* der Aktionen an. Sie können diesen Wert ändern, indem Sie die gewünschte Aktion auswählen und dann die Eigenschaft im Objektinspektor bearbeiten.

Die zweite Liste enthält die Aktionsnamen. Sie können diesen Wert ändern, indem Sie eine Aktion auswählen und dann ihre Eigenschaft *Name* im Objektinspektor bearbeiten.

Klicken Sie mit der rechten Maustaste in den unteren Bereich des Aktionsmanager-Editors, um das zugehörige Kontextmenü anzuzeigen. Es enthält die folgenden Einträge:

Befehl Beschreibung

Neue Aktion Fügt eine neue (nicht vordefinierte) Aktion in den Aktionsmanager-Editor ein. Im Objektinspektor können Sie deren Eigenschaften bearbeiten.

Neue Standardaktion Anzeigen des Dialogfeldes Standardaktionen, in dem eine vordefinierte Aktion gewählt werden kann.

Nach oben Verschieben der ausgewählten Aktion nach oben in der Liste.

Nach unten Verschieben der ausgewählten Aktion nach unten in der Liste.

Ausschneiden Ausschneiden der ausgewählten Aktion in die Zwischenablage.

Kopieren Kopieren der ausgewählten Aktion in die Zwischenablage.

Einfügen Einfügen einer Aktion aus der Zwischenablage oberhalb des ausgewählten Eintrags.

Löschen Löschen der ausgewählten Aktion.

Alles markieren Markieren aller Aktionen in der Liste.

Registerseite Optionen

Diese Registerseite ist in zwei Abschnitte unterteilt. Der obere Abschnitt, Angepasste Menüs und Symbolleisten, enthält das Kontrollkästchen Zuletzt verwendete Einträge zuerst im Menü anzeigen, das festlegt, wie Menüeinträge angezeigt werden, und die Schaltfläche Verwendete Daten zurücksetzen, die *TActionBands*-Komponenten der Anwendung auf ihre Ausgangseinstellungen zurücksetzt.

Der zweite Abschnitt dieser Registerkarte enthält ein Kontrollkästchen, das bewirkt, dass große Symbole auf *TActionBands*-Komponenten angezeigt werden. Ein weiteres Kontrollkästchen steuert die Anzeige von Kurzhinweisen auf Symbolleisten. Wenn dieses Kontrollkästchen markiert ist, werden mit dem dritten Kontrollkästchen der Registerkarte in den Kurzhinweisen Tastenkürzel angezeigt. Außerdem können Sie auf dieser Registerseite noch die Art der Animation festlegen, die beim Öffnen von Menüs verwendet werden soll.

2.267 Aktionslisten-Editor

Verwenden Sie den Aktionslisten-Editor, um während des Entwurfs Aktionen zu einer `TActionList`-Komponente hinzufügen.

Den Aktionslisten-Editor öffnen

Zum Anzeigen des Aktionslisten-Editors markieren Sie das `TActionList`-Objekt und doppelklicken anschließend auf die Komponente, oder klicken Sie mit der rechten Maustaste und wählen Sie **Editor für Aktionslisten**.

Symbolleiste

Am oberen Rand des Editors befindet sich eine Symbolleiste mit vier Schaltflächen. Dies sind:

Schaltfläche

Beschreibung

Neue Aktion Einfügen einer neuen Aktion in die Liste. Durch Klicken auf den Dropdown-Pfeil neben der Schaltfläche können Sie wählen, ob eine neue selbstdefinierte oder eine vordefinierte Standardaktion hinzugefügt wird. Wenn Sie Standardaktion wählen, wird ein Dialogfeld mit einer Liste der Standardaktionen geöffnet.

Löschen Löschen der aktuell ausgewählten Aktion.

Pfeilschaltflächen Ändern der Position der aktuell ausgewählten Aktion in der Liste.

Wenn Sie mit der rechten Maustaste in der Symbolleiste klicken, wird ein lokales Menü geöffnet. Es enthält folgenden Eintrag:

Befehl Beschreibung

Textlabel Ein-/Ausblenden der Beschriftungen für die Schaltflächen in der Symbolleiste.

Listenfelder

Der untere Teil des Aktionslisten-Editors enthält zwei Listenfelder. Die erste Liste gibt den Wert der Eigenschaft `Category` der Aktionen an. Sie können diesen Wert ändern, indem Sie die gewünschte Aktion auswählen und dann die Eigenschaft im Objektinspektor bearbeiten.

Die zweite Liste enthält die Aktionsnamen. Sie können diesen Wert ändern, indem Sie eine Aktion auswählen und dann ihre Eigenschaft `Name` im Objektinspektor bearbeiten.

Klicken Sie mit der rechten Maustaste in den unteren Bereich des Aktionslisten-Editors, um das zugehörige Kontextmenü anzuzeigen. Es enthält die folgenden Einträge:

Befehl Beschreibung

Neue Aktion Hinzufügen einer neuen (nicht vordefinierten) Aktion zur Aktionsliste. Die Eigenschaften der Aktion können im Objektinspektor geändert werden.

Neue Standardaktion Anzeigen des Dialogfeldes Standardaktionen, in dem eine vordefinierte Aktion gewählt werden kann.

Nach oben Verschieben der ausgewählten Aktion nach oben in der Liste.

Nach unten Verschieben der ausgewählten Aktion nach unten in der Liste.

Ausschneiden Ausschneiden der ausgewählten Aktion in die Zwischenablage.

Kopieren Kopieren der ausgewählten Aktion in die Zwischenablage.

Einfügen Einfügen einer Aktion aus der Zwischenablage oberhalb des ausgewählten Eintrags.

Löschen Löschen der ausgewählten Aktion.

Alles markieren Markieren aller Aktionen in der Liste.

Bereichsbeschreibungen Ein-/Ausblenden der Listenbeschriftungen.

Symbolleiste Ein-/Ausblenden der Symbolleiste.

2.268 Seite hinzufügen (Dialogfeld)

Verwenden Sie das Dialogfeld **Seite hinzufügen**, um einer Notebook- oder TabbedNotebook-Komponente Arbeitsblattseiten hinzuzufügen.

Um das Dialogfeld zu öffnen, klicken Sie im Arbeitsblatt-Editor auf **Hinzufügen**. Seite hinzufügen zur Verfügung: **Seitenname**

Geben Sie den Namen der Arbeitsblattseite ein. Seitennamen dürfen nicht länger als 255 Zeichen sein.

Hilfekontext

Geben Sie die Kontextnummer für die Arbeitsblattseite ein. Diese Nummer ist von Bedeutung, wenn Sie für die einzelnen Seiten des Arbeitsblattes kontextbezogene Hilfe implementieren wollen. Der Hilfekontext ist optional.

2.269 Auflistungs-Editor

Der Kollektionseditor dient zum Bearbeiten von Einträgen für Kollektionen. Ein Kollektionsobjekt ist ein Nachkomme von TCollection. In den Dialogfeldern werden Informationen über die Einträge in der Kollektion angezeigt. Sie können Einträge hinzufügen, entfernen oder neu anordnen. Für einige Kollektionen stehen zusätzliche Schaltflächen für weitere Manipulationen der Liste zur Verfügung.

Die Einträge in den Listenfeldern dieser Dialogfelder können mit der Maus markiert werden. Die Eigenschaften und Ereignisse eines markierten Eintrags werden im Objektinspektor eingestellt.

Den Kollektionseditor öffnen

Platzieren Sie zum Anzeigen des Kollektionseditors zuerst die Komponente im Formular, die die Kollektion verwendet. Markieren Sie im Objektinspektor die durch die Kollektion implementierte Eigenschaft (in der obigen Tabelle in Klammern dargestellt), und klicken Sie auf den Ellipsen-Schalter. Bei einigen Komponenten lässt sich das Dialogfeld zum Bearbeiten der Kollektion auch anzeigen, indem Sie die Komponente mit der rechten Maustaste anklicken und aus dem Kontextmenü den entsprechenden Editor auswählen.

Dialogfeldoptionen

Im Kollektionseditor stehen die folgenden Optionen zur Verfügung.

Einträge

Die Liste der Einträge zeigt die Eigenschaften für jeden Eintrag der Kollektion an, die in der dritten Spalte der obigen Tabelle aufgeführt sind. Die Eigenschaften des markierten Eintrags werden im Objektinspektor angezeigt und bearbeitet.

Schalter Hinzufügen

Fügt der Kollektion einen neuen Eintrag hinzu. Sie können den Eintrag markieren und seine Parameter im Objektinspektor bearbeiten.

Löschen (Schaltfläche)

Entfernt den markierten Eintrag aus der Kollektion.

Nach oben/Nach unten (Schaltflächen)

Ändert die Reihenfolge der Einträge. Bei den meisten Kollektionen wird damit die Reihenfolge festgelegt, in der die Einträge angezeigt oder von dem Objekt, das die Kollektion enthält, verwendet werden.

Alle Felder hinzufügen (nur TDBGridColumns)

Fügt für jedes Feld der Datenmenge, mit der das datensensitive Gitter verbunden ist, eine Spalte hinzu. Diese Schaltfläche kann nur aktiviert werden, wenn das datensensitive Gitter mit einer aktiven Datenmenge verbunden ist.

Vorgabe (nur TDBGridColumns)

Stellt die Standardeigenschaften (übernommen von der Feldkomponente) der aktuell markierten Spalte wieder her. Diese Schaltfläche kann nur aktiviert werden, wenn die aktuell markierte Spalte mit einem Feld verknüpft (d. h. die Eigenschaft `FieldName` gesetzt) ist.

Aus Dictionary lesen (nur TCheckConstraints)

Fügt für jede Beschränkung auf Datensatzebene im Daten-Dictionary ein CheckConstraint-Objekt hinzu. Die Eigenschaft `ImportedConstraint` jedes CheckConstraint-Objekts wird auf die Beschränkung im Dictionary gesetzt.

Beispiele für Kollektionselemente

Die folgende Tabelle enthält Beispiele für Kollektionselemente. Die Liste ist nicht vollständig, da ständig neue Kollektionen

hinzugefügt werden.

Kollektion	Elementtyp	Angezeigte Eigenschaften	Verwendung
TAggregates	TAggregates	Aggregation	Sie können den Editor benutzen, um zur Entwurfszeit Aggregationsfelder in einen Client-Datensatz einzufügen. Bei der Definition von Aggregatfeldern zur Entwurfszeit erzeugt der Editor automatisch das erforderliche TAggregate-Objekt.
TCheckConstraints	TCheckConstraint	ImportedConstraint oder, wenn kein ImportedConstraint leer ist, CustomConstraint	Stellt eine Einschränkung auf Datensatzebene für die Daten einer Datenmenge dar. (Eigenschaft Constraints)
TCoolBands	TCoolBand	Text	Stellt eine Gruppe von Abschnitten einer CoolBar-Komponente dar. (Eigenschaft Bands)
TDBGridColumns	TColumn	FieldName	Stellt die Feldbindungs- und -anzeige-Eigenschaften einer Spalte in einem datensensitiven Gitter dar. (Eigenschaft Columns).
TIndexDefs	TIndexDefs	IndexDefs	Beschreibt einen Index in einer Datenbanktabelle.
THeaderSections	THeaderSection	Text	Stellt die Anzeige-Eigenschaften der Abschnitte eines HeaderControl-Objekts dar. (Eigenschaft Sections)
TListColumns	TListColumn	Titel	Stellt die Spalten einer reportartigen ListView-Komponente dar. (Eigenschaft Columns).
TStatusPanels	TStatusBar	Text	Stellt die einzelnen Bereiche einer StatusBar-Komponente dar. (Eigenschaft Panels)
TWebActionItems	TWebActionItem	Name, Enabled und Default	Stellt einen Aktionseintrag dar, der die Antwort auf eine HTTP-Anforderungsbotschaft für einen Web-Dispatcher oder ein Web-Modul erzeugt. (Eigenschaft Actions)

2.270 Seite bearbeiten (Dialogfeld)

Verwenden Sie das Dialogfeld **Seite bearbeiten**, um vorhandene Arbeitsblätter einer Notebook- oder TabbedNotebook-Komponente zu bearbeiten.

Um das Dialogfeld zu öffnen, klicken Sie im Arbeitsblatt-Editor auf **Bearbeiten**.

Optionen des Dialogfeldes Seite bearbeiten

Folgende Optionen stehen zur Wahl:

Seitename

Geben Sie den Namen der Arbeitsblattseite ein. Seitennamen dürfen nicht länger als 255 Zeichen sein.

Hilfekontext

Geben Sie die Kontextnummer für die Arbeitsblattseite ein. Diese Nummer ist von Bedeutung, wenn Sie für die einzelnen Seiten des Arbeitsblattes kontextbezogene Hilfe implementieren wollen. Der Hilfekontext ist optional.

2.271 IconView-Eintragseditor

Verwenden Sie den IconView-Eintragseditor, um während des Entwurfs Einträge zu einer Symbolansicht (TIconView) hinzuzufügen oder aus ihr zu entfernen. Für jedes Element können Sie den Titel sowie die Bildnummer angeben.

Den IconView-Eintragseditor öffnen

So öffnen Sie den IconView-Eintragseditor:

- Markieren Sie ein TIconView-Objekt im Formular und klicken Sie im Objektinspektor auf die Ellipsen-Schaltfläche neben dem Wert der Eigenschaft Items oder
- doppelklicken Sie im Formular auf ein TIconView-Objekt oder
- klicken Sie im Formular mit der rechten Maustaste auf ein TIconView-Objekt und wählen Sie im angezeigten Kontextmenü den Befehl **Eintragseditor**.

Verwendung des IconView-Eintragseditors

Der IconView-Eintragseditor enthält das Gruppenfeld **Einträge**, in dem sich das Listenfeld **Einträge** sowie die Schaltflächen **Neuer Eintrag** und **Löschen** befinden. Wenn Sie dem Formular ein TIconView-Objekt hinzufügen, ist das Listenfeld **Einträge** zunächst leer, und die Schaltfläche **Löschen** sowie die Item-Eigenschaften auf der rechten Seite sind deaktiviert. Wenn Sie die Eigenschaften des ausgewählten Eintrags ändern, wird die Schaltfläche **Übernehmen** aktiviert, mit der die neuen Einstellungen sofort übernommen werden können.

Der IconView-Eintragseditor enthält außerdem das Gruppenfeld **Eigenschaften des Eintrags**, mit dem die Eigenschaften des aktuell in der Liste **Einträge** ausgewählten IconView-Eintrags festgelegt werden können. Dazu stehen Ihnen die Eingabefelder **Titel** und **Bildindex** zur Verfügung.

Einträge (Gruppenfeld)

Einträge (Listenfeld) Zeigt die IconView-Einträge an. Die Einträge sind in der Eigenschaft Items des TIconView-Objekts enthalten.

Neuer Eintrag (Schaltfläche) Ermöglicht dem TIconView-Objekt neue Einträge hinzuzufügen.

Löschen (Schaltfläche) Ermöglicht das Löschen des ausgewählten Eintrags aus der Liste **Einträge**.

Im Gruppenfeld **Einträge** können Sie IconView-Einträge und -Untereinträge erstellen oder entfernen. Um einen neuen Eintrag hinzuzufügen, klicken Sie auf **Neuer Eintrag**. Für jeden Eintrag wird in der Liste **Einträge** eine Standardbeschriftung angezeigt. Die Eigenschaften eines Eintrags können mit den Einstellungen im Gruppenfeld **Eigenschaften des Eintrags** geändert werden. Wenn Sie einen neuen Eintrag erstellen oder einen vorhandenen auswählen, wird die Schaltfläche **Neuer Untereintrag** aktiviert, so dass Sie Einträge in anderen Einträgen des TIconView-Objekts verschachteln können. Enthält die Liste **Einträge** Elemente, ist auch die Schaltfläche **Löschen** aktiviert. Um einen Eintrag zu löschen, wählen Sie ihn in der Liste aus und klicken auf **Löschen**.

Eigenschaften des Eintrags (Gruppenfeld)

Titel Legt den Namen für den Eintrag fest, der im Listenfeld Einträge ausgewählt ist. Dieser Name wird in der Symbolansicht angezeigt.

Bildindex Ermöglicht die Angabe des Index der Grafik, die neben dem gewählten IconView-Eintrag angezeigt werden soll. Die Einträge sind in der Eigenschaft Images des TIconView-Objekts enthalten.

Im Gruppenfeld Eigenschaften des Eintrags definieren Sie die Eigenschaften eines ausgewählten Elements. Geben Sie seinen Namen in das Eingabefeld **Titel** ein. Der Eintrag wird dann in der Liste **Einträge** entsprechend geändert.

Um eine Grafik links neben einem nicht ausgewählten Eintrag anzuzeigen, geben Sie den Index der gewünschten Grafik in das

Eingabefeld **Bildindex** ein. Soll keine Grafik angezeigt werden, setzen Sie **Bildindex** auf 0 (Standardwert).

2.272 Bilderlisten-Editor

Verwenden Sie den Bilderlisten-Editor, um während des Entwurfs Bitmaps und Symbole zu einer `TImageList`-Komponente hinzuzufügen.

Mithilfe der Schaltfläche **Übernehmen** können Sie Ihre Änderungen speichern, ohne den Editor zu beenden. Wenn Sie auf **OK** klicken, werden die Änderungen gespeichert und das Dialogfeld geschlossen. Die Schaltfläche **Übernehmen** ist besonders hilfreich, da nach Verlassen des Editors keine Änderungen mehr durchgeführt werden können.

Zum Anzeigen des Bilderlisten-Editors markieren Sie das `TImageList`-Objekt und doppelklicken Sie anschließend auf die Komponente, oder klicken Sie mit der rechten Maustaste und wählen Sie **Bilderlisten-Editor**.

Ausgewählte Grafik

In diesem Feld wird die aktuell ausgewählte Grafik angezeigt. Sie können in der Bilderliste am unteren Rand des Dialogfeldes eine andere Grafik auswählen. Wenn eine Grafik ausgewählt ist, kann sie aus der Liste entfernt werden. Wurde die Grafik in der aktuellen Editorsitzung hinzugefügt, können ihre Eigenschaften mit den anderen Steuerelementen geändert werden. Jedoch werden diese Einstellungen nach dem Schließen des Editors fixiert und können später nicht mehr geändert werden.

Transparent

Verwenden Sie diese Dropdown-Liste, um die Farbe festzulegen, die beim transparenten Zeichnen der Grafik für die Maske verwendet wird. Als Standardwert wird die Farbe des äußersten linken Pixels in der untersten Zeile verwendet. Sie können die Farbe auch ändern, indem Sie direkt auf das gewünschte Pixel klicken.

Wenn eine Grafik eine Transparentfarbe hat, werden alle Pixel mit dieser Farbe transparent angezeigt. Auf diese Weise ist an diesen Stellen der Hintergrund der Grafik zu sehen.

Bei einem Symbol hat **Transparent** den Wert `cNone` und kann nicht geändert werden (Symbole sind bereits maskiert).

Füllfarbe

Verwenden Sie diese Dropdown-Liste, um eine Farbe anzugeben, die um den Rand der Grafik hinzugefügt wird, wenn diese kleiner als die Bilderlistenkomponente ist (Eigenschaften `Height` und `Width`).

Das Steuerelement ist deaktiviert, wenn die Grafik die Bilderliste vollständig ausfüllt (d. h. mindestens so groß wie die Eigenschaften `Height` und `Width` ist). Auch bei Symbolen steht es nicht zur Verfügung, da Symbole Masken entsprechen, bei denen alle Ränder transparent sind.

Optionen

Mithilfe der Optionsfeldgruppe **Optionen** können Sie angeben, wie die Grafik in der Bilderliste angezeigt wird, wenn sie nicht genau in diese passt. Die Optionsfelder sind bei Symbolen deaktiviert.

EinstellungBeschreibung

Zuschneiden Die Grafik wird an den Rändern der Bilderliste abgeschnitten.

Dehnen Die Grafik wird an die Größe der Bilderliste angepasst.

Zentrieren Die Grafik wird in der Bilderliste zentriert. Ist die Grafik größer als die Komponente, wird sie abgeschnitten.

Bilder

Dieses Feld enthält eine Vorschau der Grafiken in der Bilderliste sowie Steuerelemente für das Hinzufügen und Entfernen von Bildern. Jede Grafik wird in einem Bereich von 24x24 Pixel angezeigt, damit mehrere Bilder besser zu sehen sind. Unter den einzelnen Bildern befindet sich eine Beschriftung, aus der ihre Position in der Bilderliste (beginnend bei 0) zu erkennen ist. Sie können die Position einer Grafik ändern, indem Sie diesen Wert ändern oder das Bild an die gewünschte Position ziehen.

Hinzufügen

Mit dieser Schaltfläche öffnen Sie das Dialogfeld **Bilder hinzufügen**, in dem Sie die gewünschten Bilder bzw. Symbole auswählen können. Die Grafiken werden nach dem Hinzufügen hervorgehoben und erhalten die nächsten sequentiellen Werte in der Bilderliste.

Ist ein Bitmap um ein Vielfaches größer als die Komponente, werden Sie in einem Dialogfeld gefragt, ob es in mehrere Grafiken aufgeteilt werden soll. Dies ist speziell für Symbolleisten-Bitmaps hilfreich, die normalerweise aus mehreren aufeinanderfolgenden kleinen Grafiken bestehen und als größeres Bitmap gespeichert sind.

Entfernen

Mit dieser Schaltfläche können Sie die ausgewählte Grafik aus der Bilderliste entfernen. Die verbleibenden Grafiken werden anschließend neu angeordnet, so dass sie wieder eine fortlaufende Liste bilden.

Löschen

Mit dieser Schaltfläche können Sie alle Grafiken aus der Bilderliste entfernen.

Exportieren

Mit dieser Schaltfläche können Sie die ausgewählte Grafik in einer Datei speichern. Das Bild wird dabei mit seinem aktuellen Status gespeichert (einschließlich Abschneiden oder Dehnen).

2.273 ListView Items Editor

Verwenden Sie den ListView-Eintragseditor, um während des Entwurfs die Einträge in einer Listenansicht zu bearbeiten. Sie können Einträge und Untereinträge hinzufügen oder entfernen sowie Beschriftung und Bildindex der verschiedenen Einträge festlegen.

Zum Anzeigen des ListView-Eintragseditors wählen Sie das TListView-Objekt aus, und doppelklicken Sie im Objektinspektor auf den Wert der Eigenschaft Items.

Verwenden des Editors

Der Eintragseditor enthält das Gruppenfeld **Einträge**, in dem sich das Listenfeld **Einträge** sowie die Schaltflächen **Neuer Eintrag**, **Neuer Untereintrag** und **Löschen** befinden. Beim ersten Öffnen des Editors ist das Listenfeld leer, und die Schaltflächen **Neuer Untereintrag** und **Löschen** sind deaktiviert. Wenn Sie die Eigenschaften des ausgewählten Eintrags ändern, wird die Schaltfläche **Übernehmen** aktiviert, mit der die neuen Einstellungen sofort übernommen werden können.

Im Editor ist auch das Gruppenfeld **Eigenschaften des Eintrags** vorhanden, mit dem die Eigenschaften des aktuell in der Liste ausgewählten Eintrags festgelegt werden können. Dazu stehen Ihnen die Eingabefelder **Titel** und **Bildindex** zur Verfügung.

Einträge (Gruppenfeld)

Hier können Sie Einträge und Untereinträge erstellen oder entfernen. Um einen neuen Eintrag hinzuzufügen, klicken Sie auf **Neuer Eintrag**. Für jeden Eintrag wird in der Liste **Einträge** eine Standardbeschriftung angezeigt. Die Eigenschaften eines Eintrags können mit den Einstellungen im Gruppenfeld **Eigenschaften des Eintrags** geändert werden. Wenn Sie einen neuen Eintrag erstellen oder einen vorhandenen auswählen, wird die Schaltfläche **Neuer Untereintrag** aktiviert, so dass Sie Einträge in anderen Elementen verschachteln können. Enthält die Liste **Einträge** Elemente, ist auch die Schaltfläche **Löschen** aktiviert. Um einen Eintrag zu löschen, wählen Sie ihn in der Liste aus und klicken auf **Löschen**.

Eigenschaften des Eintrags (Gruppenfeld)

Im Gruppenfeld Eigenschaften des Eintrags definieren Sie die Eigenschaften eines ausgewählten Elements. Geben Sie seinen Namen in das Eingabefeld **Titel** ein. Der Eintrag wird dann in der Liste **Einträge** entsprechend geändert.

Um eine Grafik links neben einem nicht ausgewählten Eintrag anzuzeigen, geben Sie den Index der gewünschten Grafik in das Eingabefeld **Bildindex** ein. Soll keine Grafik angezeigt werden, setzen Sie **Bildindex** auf -1 (Standardwert).

2.274 Standardaktionsklassen (Dialogfeld)

Verwenden Sie das Dialogfeld **Standardaktionsklassen**, um der Aktionsliste eine vordefinierte Standardaktion hinzuzufügen. Standardaktionen führen häufig benötigte Aufgaben durch, z. B. Navigieren in Datenmengen, Verwalten der Fenster einer MDI-Anwendung oder Arbeiten mit der Windows-Zwischenablage. Jede Standardaktion führt bei ihrem Aufruf eine bestimmte Funktion aus und aktiviert oder deaktiviert bei Bedarf die beteiligten Steuerelemente.

Wählen Sie die gewünschte Aktion aus der Liste, und klicken Sie auf **OK**. Eine Beschreibung der vordefinierten Aktionsklassen finden Sie unter Vordefinierte Aktionsklassen.

2.275 String-Listen-Editor

Verwenden Sie den String-Listen-Editor zur Entwurfszeit zum Hinzufügen, Bearbeiten, Laden und Speichern von Strings in beliebigen als *TStrings* deklarierten Eigenschaften.

So öffnen Sie den Editor:

1. Fügen Sie eine Komponente in das Formular ein, die eine Stringliste verwendet.
2. Markieren Sie die Komponente und führen Sie eine der folgenden Aktionen durch:
 - Klicken Sie bei Eigenschaften des Typs *TStrings* in der Wertespalte auf die Ellipsen-Schaltfläche (z.B. bei der Eigenschaft *Items* der Komponente *ComboBox*).
 - Doppelklicken Sie bei den Eigenschaften des Typs *TStrings* in der Wertespalte auf (*TStrings*).

Anmerkung: Wenn die Eigenschaft eine Werteliste ist, wird der Wertlisten-Editor angezeigt.

Quelltext-Editor (Schaltfläche)

Um die Liste in Text zu konvertieren, klicken Sie die Schaltfläche Quelltext-Editor an. Die Liste wird auf einer eigenen Registerseite im Editor angezeigt. Hier können Sie sie mit den Standard-Editierbefehlen bearbeiten.

Das Kontextmenü des String-Listen-Editors

Das Kontextmenü des String-Listen-Editors (klicken Sie im Editor mit der rechten Maustaste) enthält folgende Optionen:

Laden

Klicken Sie auf Laden, um das Dialogfeld String-Liste laden anzuzeigen, in dem Sie eine existierende Datei auswählen können, die in den String-Listen-Editor eingelesen werden soll.

Speichern

Klicken Sie auf Speichern, um die aktuelle String-Liste in eine Datei zu schreiben. Das Dialogfeld String-Liste speichern wird geöffnet, in dem Sie ein Verzeichnis und einen Dateinamen angeben können.

Siehe auch

Wertlisten-Editor (siehe Seite 627)

String-Liste laden (siehe Seite 635)

String-Liste speichern (siehe Seite 642)

2.276 TreeView-Eintragseditor

Verwenden Sie den TreeView-Eintragseditor, um während des Entwurfs Einträge zu einem Baumdiagramm hinzuzufügen, vorhandene Einträge zu löschen oder Grafiken von der Festplatte in eine Komponente zu laden. Sie können den einzelnen Einträgen Text zuweisen und Werte für Bildindex, Ausgewählt-Index und Statusindex angeben.

Zum Anzeigen des TreeView-Eintragseditors wählen Sie das TTreeView-Objekt aus, und doppelklicken Sie im Objektinspektor auf den Wert der Eigenschaft Items.

Verwenden des Editors

Der Eintragseditor enthält das Gruppenfeld **Einträge**, in dem sich das Listenfeld **Einträge** sowie die Schaltflächen **Neuer Eintrag**, **Neuer Untereintrag**, **Löschen** und **Laden** befinden. Beim ersten Öffnen des Editors ist das Listenfeld leer, und die Schaltflächen **Neuer Untereintrag** und **Löschen** sind deaktiviert. Wenn Sie die Eigenschaften des ausgewählten Eintrags ändern, wird die Schaltfläche **Übernehmen** aktiviert, mit der die neuen Einstellungen sofort übernommen werden können.

Der Editor enthält das Gruppenfeld **Untereinträge** mit dem Listenfeld **Untereinträge** und den Schaltflächen **Untereintrag hinzufügen** und **Löschen**.

Im Editor ist auch das Gruppenfeld **Eigenschaften des Eintrags** vorhanden, mit dem die Eigenschaften des aktuell in der Liste ausgewählten Eintrags festgelegt werden können. Dazu stehen Ihnen die Eingabefelder **Text**, **Bildindex**, Ausgewählt-Index und **Statusindex** zur Verfügung.

Einträge (Gruppenfeld)

Hier können Sie Einträge und Untereinträge erstellen, laden oder entfernen. Mithilfe der Schaltfläche **Laden** kann eine Gruppe vorhandener Einträge von der Festplatte eingelesen werden. Um einen neuen Eintrag hinzuzufügen, klicken Sie auf **Neuer Eintrag**. Für jeden Eintrag wird in der Liste **Einträge** ein Standardtext angezeigt. Die Eigenschaften eines Eintrags können mit den Einstellungen im Gruppenfeld **Eigenschaften des Eintrags** geändert werden.

Wenn Sie einen neuen Eintrag erstellen oder einen vorhandenen auswählen, wird die Schaltfläche **Neuer Untereintrag** aktiviert, so dass Sie Einträge in anderen Elementen verschachteln können. Enthält die Liste **Einträge** Elemente, ist auch die Schaltfläche **Löschen** aktiviert. Um einen Eintrag zu löschen, wählen Sie ihn in der Liste aus und klicken auf **Löschen**.

Eigenschaften des Eintrags (Gruppenfeld)

Im Gruppenfeld Eigenschaften des Eintrags definieren Sie die Eigenschaften eines ausgewählten Elements. Geben Sie seinen Text in das Eingabefeld **Text** ein. Der Eintrag wird dann in der Liste **Einträge** entsprechend geändert.

Um eine Grafik links neben einem nicht ausgewählten Eintrag anzuzeigen, geben Sie den Index der gewünschten Grafik in das Eingabefeld **Bildindex** ein. Soll keine Grafik angezeigt werden, setzen Sie **Bildindex** auf -1 (Standardwert).

Um eine zusätzliche Grafik links neben einem Eintrag anzuzeigen, geben Sie den Index der gewünschten Grafik in das Eingabefeld **Ausgewählt-Index** ein. Die Indexzählung beginnt bei 0. Soll keine Grafik angezeigt werden, setzen Sie **Ausgewählt-Index** auf -1 (Standardwert).

Um eine zusätzliche Grafik links neben einem Eintrag anzuzeigen, geben Sie den Index der gewünschten Grafik in das Eingabefeld Statusindex ein. Dieser Wert entspricht dem Index der Eigenschaft StatelImages der ListView-Komponente. Die Indexzählung beginnt bei 0. Soll keine Grafik angezeigt werden, setzen Sie Statusindex auf -1 (Standardwert).

2.277 Wertlisten-Editor

Verwenden Sie den Wertlisten-Editor, um zur Entwurfszeit Namen-Werte-Paare für jede beliebige Eigenschaft, die als ein *TStrings*-Objekt deklariert wurde, hinzuzufügen, zu bearbeiten, zu laden und zu speichern.

So öffnen Sie den Editor:

1. Fügen Sie eine Komponente in das Formular ein, die eine Stringliste verwendet.
2. Markieren Sie die Komponente und führen Sie eine der folgenden Aktionen durch:
 - Klicken Sie bei den Eigenschaften des Typs *TStrings* in der Wertespalte auf die Ellipsen-Schaltfläche.
 - Doppelklicken Sie bei den Eigenschaften des Typs *TStrings* in der Wertespalte auf (*TStrings*).

Um der Wertliste Einträge hinzuzufügen, geben Sie den Namen des Eintrags in der Spalte **Schlüssel** und dessen Wert in der Spalte **Wert** ein. Wenn Sie OK anklicken, wird die Stringliste gespeichert.

Um die Liste in Text zu konvertieren, klicken Sie die Schaltfläche Quelltext-Editor an. Die Liste wird auf einer eigenen Seite im Editor angezeigt. Hier können Sie die Liste mit Hilfe der Editerbefehle bearbeiten.

Das Kontextmenü des Wertlisten-Editors

Das Kontextmenü des Wertlisten-Editors (klicken Sie im Editor mit der rechten Maustaste) enthält folgende Optionen:

Laden

Klicken Sie auf Laden, um das Dialogfeld String-Liste laden anzuzeigen, in dem Sie eine vorhandene Datei auswählen können, die in den Wertlisten-Editor eingelesen werden soll.

Speichern

Klicken Sie auf Speichern, um die aktuelle String-Liste in eine Datei zu schreiben. Das Dialogfeld String-Liste speichern wird geöffnet, in dem Sie ein Verzeichnis und einen Dateinamen angeben können.

Siehe auch

String-Liste laden (siehe Seite 635)

String-Liste speichern (siehe Seite 642)

2.278 Eingabemasken-Editor

Verwenden Sie den Eingabemasken-Editor zum Definieren eines Eingabefeldes, das dem Benutzer nur ein bestimmtes Format erlaubt und das nur gültige Zeichen akzeptiert. So können Sie beispielsweise für Telefonnummern ein Eingabefeld definieren, das nur numerische Eingaben zuläßt. Versucht der Benutzer, einen Buchstaben einzugeben, akzeptiert ihn die Anwendung nicht.

Im Eingabemasken-Editor kann die Eigenschaft **EditMask** der MaskEdit-Komponente geändert werden.

Den Eingabemasken-Editor öffnen

So öffnen Sie den Eingabemasken-Editor:

1. Fügen Sie dem Formular eine MaskEdit-Komponente hinzu.
2. Markieren Sie die Komponente und führen Sie eine der folgenden Aktionen durch:
 - Klicken Sie auf die Ellipsen-Schaltfläche in der Wertespalte der Eigenschaft **EditMask**.
 - Doppelklicken Sie in der Wertespalte der Eigenschaft **EditMask**.

Eingabemaske

Definieren Sie Ihre eigenen Masken für Eingabefelder. Eine Liste der zulässigen Zeichen finden Sie bei der Beschreibung der Eigenschaft **EditMask**.

Die Maske besteht aus drei durch Semikolons voneinander abgeteilten Feldern. Diese drei Felder sind:

- Die eigentliche Maske (Sie können die vordefinierten Masken verwenden oder eine eigene Maske erstellen).
- Das Zeichen, das festlegt, ob Literalzeichen in der Maske als Teil der Daten gespeichert werden.
- Das Zeichen, das in der Maske zur Darstellung eines Leerzeichens verwendet wird.

Leerzeichen als

Legen Sie ein Zeichen fest, das in der Maske für ein Leerzeichen stehen soll. Leerzeichen in der Maske sind Bereiche, die vom Benutzer Eingaben erwarten.

Dieses Eingabefeld ändert das dritte Feld Ihrer Eingabemaske.

Literalzeichen speichern

Aktivieren Sie dieses Kontrollkästchen, damit die Literalzeichen der Eingabemaske als Teil der Daten gespeichert werden. Diese Option wirkt sich nur auf die Eigenschaft **Text** der MaskEdit-Komponente aus. Wenn Sie Daten mit der Eigenschaft **EditText** speichern, werden die Literalzeichen automatisch berücksichtigt.

Mit dem Kontrollkästchen **Literalzeichen speichern** wird das zweite Feld in der Eingabemaske festgelegt.

Testeingabe

Im Eingabefeld **Testeingabe** können Sie die Funktion Ihrer Maske überprüfen. Das Eingabefeld zeigt die Eingabemaske so an, wie sie im Formular erscheinen wird.

Beispielmasken

In dieser Liste können Sie eine vordefinierte Maske auswählen, die in der MaskEdit-Komponente verwendet werden soll. Die gewählte vordefinierte Maske wird in das Feld **Eingabemaske** übernommen. Gleichzeitig wird im Feld **Testeingabe** eine Vorschau der Maske angezeigt. Mithilfe der Schaltfläche **Masken** können Sie eine Liste mit Maskendateien für Ihr Land anzeigen.

Schaltfläche Masken

Klicken Sie auf **Masken**, um das Dialogfeld **Maskendatei öffnen** anzuzeigen. Hier können Sie eine Datei auswählen, welche die im Listenfeld **Beispielmasken** gezeigten Masken enthält.

2.279 Template einfügen

Kontextmenü Menü-Designer ▶ Aus Template einfügen

Verwenden Sie dieses Dialogfeld dazu, ein Menü mit einem vordefinierten Template einzufügen.

Menü Bearbeiten	Fügt das Standard-Bearbeitungsmenü mit den Elementen Rückgängig, Wiederholen, Ausschneiden, Kopieren, Einfügen, Inhalt einfügen, Suchen, Ersetzen, Gehe zu, Beziehungen und Objekte ein.
Menü Datei	Fügt das Standard-Dateimenü mit den Elementen Neu, Öffnen, Speichern, Speichern unter, Drucken, Drucker einrichten und Beenden ein.
Menü Datei (für TextEdit-Beispiel)	Fügt das Dateimenü mit den Standardelementen und auch den Menüpunkt Schließen ein.
Menü Hilfe	Fügt das Hilfemenü mit den Elementen Inhalt, Themenbezogene Hilfe, Hilfe verwenden und Info ein.
Menü Hilfe (erweitert)	Entfernt das Hilfemenü mit den Standardelementen plus weiteren Elementen wie Index, Anweisungen, Anleitungen, Tastatur und Tutorial.
MDI-Framemenü	Fügt einen Satz von Menüs ein, die dieselben Elemente wie die Menüs Datei, Bearbeiten, Windows und Hilfe (erweitert) enthalten.
Fenster	Fügt das Menü Fenster mit den Elementen Neues Fenster, Nebeneinander, Überlappend, Alle anordnen, Schließen und Anzeigen ein.

2.280 Objekt einfügen

Verwenden Sie das Dialogfeld **Objekt einfügen** zur Entwurfszeit, um ein OLE-Objekt in eine OleContainer-Komponente einzufügen. Die Komponente *OleContainer* ermöglicht es Ihnen, Anwendungen zu erzeugen, die Daten gemeinsam mit einer OLE-Server-Anwendung benutzen können. Nach Einfügen eines OLE-Server-Objekts in Ihre Anwendung können Sie mit einem Doppelklick auf die *OleContainer*-Komponente die Server-Anwendung starten.

Geben Sie an, ob Sie mit dem zugehörigen OLE-Server eine neue Datei erzeugen oder eine vorhandene Datei verwenden wollen. Wenn Sie eine vorhandene Datei benutzen, so muss diese mit einer Anwendung verknüpft sein, die als OLE-Server verwendbar ist.

Neu erstellen

Markieren Sie dieses Optionsfeld, wenn Sie eine Server-Anwendung aufrufen möchten, um ein neues OLE-Objekt zu erzeugen. Nach dem Markieren von Neu erstellen wird das Listenfeld Objekttyp angezeigt.

Aus Datei erstellen

Markieren Sie dieses Optionsfeld, wenn das OLE-Objekt bereits als Datei gespeichert wurde. Ist dieses Optionsfeld markiert, werden die Optionen Datei, Durchsuchen und Verknüpfen angezeigt.

Objekttyp

Wählen Sie eine Anwendung aus, die Sie als OLE-Server benutzen wollen. In diesem Listenfeld werden alle verfügbaren Anwendungen angezeigt, die als OLE-Server verwendet werden können. Nach der Auswahl einer Anwendung wird diese von gestartet.

Datei

Geben Sie den vollständigen Pfad der Datei ein, die Sie in Ihre Anwendung einfügen wollen. Die von Ihnen gewählte Datei muss mit einer Anwendung verknüpft sein, die als OLE-Server verwendbar ist.

Anmerkung: Diese Option ist nur verfügbar, wenn das Optionsfeld Aus Datei erstellen ausgewählt ist.

Durchsuchen

Klicken Sie auf Durchsuchen, um das Dialogfeld Durchsuchen anzuzeigen, in dem Sie eine als OLE-Server zu verwendende Datei auswählen können.

Anmerkung: Diese Option ist nur verfügbar, wenn das Optionsfeld Aus Datei erstellen ausgewählt ist.

Beziehung

Aktivieren Sie dieses Feld, um das Objekt auf dem Formular mit einer Datei zu verknüpfen. Wenn ein Objekt verknüpft ist, wird es automatisch aktualisiert, sobald die Quelldatei modifiziert wird. Ist Verknüpfen nicht aktiviert, betten Sie das Objekt ein, und Änderungen am Original spiegeln sich nicht in Ihrem Container wider.

Als Symbol

Aktivieren Sie dieses Feld, um das eingefügte Objekt auf dem Formular als Symbol darzustellen. Ist diese Option aktiviert, wird die Schaltfläche Anderes Symbol angezeigt.

Symbol ändern

Klicken Sie auf Anderes Symbol, um das entsprechende Dialogfeld zu öffnen, in dem Sie ein Symbol und eine Beschriftung für das von Ihnen in das Formular eingefügte Objekt festlegen können.

Anmerkung: Diese Option ist nur verfügbar, wenn das Optionsfeld Aus Datei erstellen ausgewählt ist.

Siehe auch

Durchsuchen ( siehe Seite 603)

2.281 Laden eines Bildes zur Entwurfszeit

Verwenden Sie den Bild-Editor, um Bilder in die grafikkompatiblen Komponenten zu laden und um ein Bild anzugeben, das zur Laufzeit das Formular repräsentieren soll.

Jede grafikkompatible Komponente hat eine Eigenschaft, die den Bild-Editor verwendet.

So laden Sie ein Bild zur Entwurfszeit:

1. Fügen Sie Ihrem Formular eine grafikkompatible Komponente hinzu (z.B. *TImage*).
2. Damit die Komponente automatisch an die Größe der Grafik angepasst wird, setzen Sie ihre Eigenschaft *AutoSize* auf **true**, bevor Sie die Grafik laden.
3. Wählen Sie im Objektinspektor die Eigenschaft aus, die den Bild-Editor verwendet.
4. Doppelklicken Sie entweder in der Wertespalte, oder wählen Sie die Ellipsen-Schaltfläche, um den Bild-Editor zu öffnen.

Um den Bild-Editor direkt von einer *TImage*-Komponente aus zu öffnen, doppelklicken Sie im Formular auf die Komponente.

1. Wählen Sie die Schaltfläche Laden, um das Dialogfeld Bild laden zu öffnen.
2. Wählen Sie die gewünschte Grafik aus und klicken Sie auf OK.

Das Bild wird im Bild-Editor angezeigt.

1. Wählen Sie OK, um die Auswahl zu bestätigen und das Dialogfeld zu schließen.
2. Das Bild erscheint in der Komponente.

Anmerkung: Wenn Sie eine Grafik in eine *TImage*-Komponente laden, können Sie veranlassen, dass die Größe der Grafik automatisch an die Komponente angepasst wird, indem Sie die Eigenschaft *Stretch* der Komponente auf **true** setzen. (*Stretch* hat keine Auswirkung auf die Größe von Symboldateien (.ICO)).

Siehe auch

Bild-Editor ( siehe Seite 640)

Bild laden ( siehe Seite 634)

2.282 Bild laden (Dialogfeld)

Verwenden Sie das Dialogfeld **Bild laden**, um ein Bild auszuwählen, das einer der grafikkompatiblen Komponenten hinzugefügt werden soll, und um ein Symbol für Ihr Formular anzugeben.

Um das Dialogfeld zu öffnen, klicken Sie im Grafikeditor auf **Laden**.

Dialogfeldoptionen

Folgende Optionen stehen im Dialogfeld **Seite hinzufügen** zur Verfügung:

Dateiname

Geben Sie den Namen der zu ladenden Datei oder die im Listenfeld Dateien als Filter zu verwendenden Platzhalterzeichen ein.

Dateien (großes Listenfeld)

Im Listenfeld Dateien werden die Dateien des aktuellen Verzeichnisses angezeigt, die zu den Platzhalterzeichen im Eingabefeld Dateiname oder zum Dateityp im Kombinationsfeld Dateityp passen.

Dateityp

Wählen Sie einen Filter zur Anzeige der verschiedenen Dateitypen. Standardmäßig werden im Listenfeld Dateien die Symboldateien (*.ICO) des aktuellen Verzeichnisses angezeigt.

Speichern in

Wählen Sie das Verzeichnis aus, dessen Inhalt Sie zu sehen wünschen. Im Listenfeld Dateien erscheinen die Dateien des aktuellen Verzeichnisses, die zu den Platzhalterzeichen im Eingabefeld Dateiname oder zum Dateityp im Kombinationsfeld Dateityp passen.

Laufwerke

Wählen Sie das aktuelle Laufwerk. Die Verzeichnisstruktur des aktuellen Laufwerks erscheint im Listenfeld Speichern in.

2.283 String-Liste laden

Verwenden Sie das Dialogfeld **String-Liste laden** zur Auswahl einer Textdatei, die in eine Eigenschaft des Typs *TStrings* geladen werden soll.

So öffnen Sie dieses Dialogfeld:

1. Öffnen Sie den String-Listen-Editor.
2. Klicken Sie mit der rechten Maustaste und wählen Sie Laden.

Optionen des Dialogfeldes String-Liste laden

Folgende Optionen stehen im Dialogfeld **String-Liste laden** zur Verfügung:

Dateiname

Geben Sie den Namen der zu ladenden Datei oder die im Listenfeld Dateien als Filter zu verwendenden Platzhalterzeichen ein.

Dateien (großes Listenfeld)

In der Dateiliste werden die Dateien des aktuellen Verzeichnisses angezeigt, die zu den Platzhalterzeichen im Eingabefeld Dateiname oder zum Dateityp aus dem Kombinationsfeld Dateityp passen.

Dateityp

Wählen Sie einen Filter zur Anzeige der verschiedenen Dateitypen. Standardmäßig werden im Listenfeld Dateien die Textdateien (*.txt) des aktuellen Verzeichnisses angezeigt.

Speichern in

Wählen Sie das Verzeichnis aus, dessen Inhalt Sie zu sehen wünschen. Im Listenfeld Dateien erscheinen die Dateien des aktuellen Verzeichnisses, die zu den Platzhalterzeichen im Eingabefeld Dateiname oder zum Dateityp im Kombinationsfeld Dateityp passen.

Laufwerke

Wählen Sie das aktuelle Laufwerk. Die Verzeichnisstruktur des aktuellen Laufwerks erscheint im Listenfeld Speichern in.

Siehe auch

[String-Listen-Editor \(siehe Seite 625\)](#)

2.284 Maskentext-Editor

Verwenden Sie den Maskentext-Editor, um Werte in eine Editiermaske einzugeben.

Verwenden Sie den Maskentext-Editor, um die Eigenschaft Text der Komponente MaskEdit zu bearbeiten.

So öffnen Sie den Maskentext-Editor:

1. Platzieren Sie eine MaskEdit-Komponente im Formular.
2. Markieren Sie die Komponente und führen Sie eine der folgenden Aktionen durch:
 - Klicken Sie entweder auf die Ellipsen-Schaltfläche in der Wertespalte für die Eigenschaft Text oder
 - Doppelklicken Sie in der Wertespalte der Eigenschaft Text.

Eingabetext

Geben Sie in dieses Feld die Ausgangswerte für die Komponente MaskEdit ein. Diese Werte können Sie zur Laufzeit überschreiben.

Editiermaske

Zeigt die Maskendefinition für die aktuelle Komponente an.

2.285 Arbeitsblatt-Editor

Verwenden Sie den Arbeitsblatt-Editor, um Seiten in einer *TabbedNotebook*- oder *Notebook*-Komponente hinzuzufügen, zu bearbeiten, zu entfernen oder neu anzuordnen. Außerdem lassen sich damit Hilfekontext-IDs für einzelne Arbeitsblattseiten hinzufügen und ändern.

Im Arbeitsblatt-Editor werden die Seiten des aktuellen Arbeitsblatts in ihrer aktuellen Reihenfolge angezeigt. Der Editor zeigt auch den mit der jeweiligen Seite verknüpften Hilfekontext an.

So öffnen Sie den Arbeitsblatt-Editor:

1. Platzieren Sie eine *TabbedNotebook*- oder *Notebook*-Komponente aus der Kategorie Win 3.1 der Tool-Palette im Formular.
2. Markieren Sie die Komponente und führen Sie eine der folgenden Aktionen durch:
 - Klicken Sie auf die Ellipsen-Schaltfläche in der Wertespalte der Eigenschaft *Pages*.
 - Doppelklicken Sie auf die Wertespalte der Eigenschaft *Pages*.

Bearbeiten

Klicken Sie auf Bearbeiten, um das Dialogfeld Seite bearbeiten zu öffnen. Hier können Sie den Seitennamen und die Hilfekontext-ID der ausgewählten Arbeitsblattseite ändern.

Hinzufügen

Klicken Sie auf Hinzufügen, um das Dialogfeld Seite hinzufügen zu öffnen. Hier können Sie eine neue Arbeitsblattseite erstellen.

Entfernen

Klicken Sie auf Löschen, um die markierte Arbeitsblattseite zu löschen.

Nach oben/Nach unten

Mit Nach oben und Nach unten können Sie die Reihenfolge der markierten Seite(n) ändern.

Siehe auch

Seite bearbeiten ( siehe Seite 618)

Seite hinzufügen ( siehe Seite 615)

2.286 Öffnen (Dialogfeld)

Verwenden Sie das Dialogfeld **Öffnen** zur Entwurfszeit, um einen Bericht in eine Report-Komponente oder eine Multimedia-Datei in eine MediaPlayer-Komponente zu laden.

So öffnen Sie das Dialogfeld **Öffnen**:

1. Platzieren Sie eine MediaPlayer-Komponente im Formular.
2. Markieren Sie die Komponente und führen Sie eine der folgenden Aktionen durch:
 - Klicken Sie entweder auf die Ellipsen-Schaltfläche in der Wertespalte einer der darunter aufgeführten Eigenschaften, oder
 - Doppelklicken Sie in der Wertespalte einer der darunter aufgeführten Eigenschaften.

Optionen des Dialogfelds Öffnen

Folgende Optionen stehen im Dialogfeld **Öffnen** zur Verfügung:

Dateiname

Geben Sie den Namen der zu ladenden Datei oder die im Listenfeld Dateien als Filter zu verwendenden Platzhalterzeichen ein.

Dateien

In der Dateiliste werden die Dateien des aktuellen Verzeichnisses angezeigt, die zu den Platzhalterzeichen im Eingabefeld Dateiname oder zum Dateityp aus dem Kombinationsfeld Dateityp passen.

Dateityp

Wählen Sie den Dateityp, den Sie zu laden wünschen. Standardmäßig werden alle Dateien des aktuellen Verzeichnisses angezeigt. Sie können die Anzeige aber auch auf Wave-Dateien, Midi-Dateien oder Videodateien unter Windows beschränken.

Speichern in

Wählen Sie das Verzeichnis aus, dessen Inhalt Sie zu sehen wünschen. Im Listenfeld **Dateien** erscheinen die Dateien des aktuellen Verzeichnisses, die zu den Platzhalterzeichen im Eingabefeld **Dateiname** oder zum Dateityp im Kombinationsfeld **Dateityp** passen.

Laufwerke

Wählen Sie das aktuelle Laufwerk. Die Verzeichnisstruktur des aktuellen Laufwerks erscheint im Listenfeld Speichern in.

2.287 Inhalte einfügen (Dialogfeld)

Verwenden Sie das Dialogfeld **Inhalte einfügen**, um ein Objekt aus der Zwischenablage von Windows in Ihren OLE-Container einzufügen.

Quelle

Zeigt den Pfad der Datei an, die Sie einfügen wollen.

Einfügen/Verknüpfung

Wählen Sie **Einfügen**, um das Objekt in das Formular einzubetten. Wenn Sie ein Objekt in ein Formular einbetten, speichert Ihre Container-Anwendung alle Informationen für das Objekt. Eine externe Datei wird nicht benötigt.

Wählen Sie **Verknüpfung**, um das Objekt mit dem Formular zu verknüpfen. Wenn Sie ein Objekt mit einem Formular verknüpfen, wird der Hauptquelltext in einer Datei gespeichert, so dass die Quelldatei beim Aktualisieren des Objekts ebenfalls aktualisiert wird.

Als

Gibt den Typ des von Ihnen eingefügten Anwendungsobjekts an. Die angegebene Anwendung ist die Quellanwendung, von der Sie das einzufügende Objekt erhalten haben.

2.288 Bild-Editor

Verwenden Sie den Bild-Editor, um ein Bild auszuwählen, das einer der grafikkompatiblen Komponenten hinzugefügt werden soll, oder um ein Symbol für Ihr Formular festzulegen.

So öffnen Sie den Bild-Editor:

1. Platzieren Sie eine grafikkompatible Komponente (z.B. *TImage*) im Formular.
2. Markieren Sie die Komponente und führen Sie eine der folgenden Aktionen durch:
3. Klicken Sie auf die Ellipsen-Schaltfläche in der Wertespalte einer der benötigten Eigenschaften (z.B. die Eigenschaft *Picture* der *TImage*-Komponente)
4. Doppelklicken Sie in der Wertespalte einer der benötigten Eigenschaften.

Anmerkung: Um den Bild-Editor aus einer Bild-Komponente heraus zu öffnen, können Sie auch im Formular auf die Komponente doppelklicken.

Befehle im Bild-Editor

Der Bild-Editor enthält die folgenden Optionen:

Laden

Zeigt das Dialogfeld Bild laden an, in dem Sie eine vorhandene Datei auswählen können, die in den Bild-Editor eingelesen werden soll. Weitere Informationen zum Laden von Bildern in den Bild-Editor finden Sie bei den Informationen zum Laden eines Bildes zur Entwurfszeit.

Speichern

Zeigt das Dialogfeld Bild speichern unter an, in dem Sie ein Verzeichnis und einen Dateinamen angeben können, unter dem das Bild gespeichert werden soll.

Löschen

Hebt die Verknüpfung zwischen dem aktuellen Bild und der markierten Komponente auf.

Siehe auch

Bild laden (siehe Seite 634)

Laden eines Bildes zur Entwurfszeit (siehe Seite 633)

Bild speichern unter (siehe Seite 641)

2.289 Bild speichern unter

Verwenden Sie das Dialogfeld Bild speichern unter, um das in den Bild-Editor geladene Bild in einer neuen Datei oder einem neuen Verzeichnis zu speichern.

Um das Dialogfeld zu öffnen, klicken Sie im Bild-Editor auf Speichern.

Optionen des Dialogfeldes Bild speichern unter

Folgende Optionen stehen im Dialogfeld Bild speichern unter zur Verfügung:

Dateiname

Geben Sie den Namen der zu ladenden Datei oder die im Listenfeld Dateien als Filter zu verwendenden Platzhalterzeichen ein.

Dateien (großes Listenfeld)

Im Listenfeld Dateien werden die Dateien des aktuellen Verzeichnisses angezeigt, die zu den Platzhalterzeichen im Eingabefeld Dateiname oder zum Dateityp im Kombinationsfeld Dateityp passen.

Dateityp

Wählen Sie einen Filter zur Anzeige der verschiedenen Dateitypen. Standardmäßig werden im Listenfeld Dateien die Symboldateien (*.ICO) des aktuellen Verzeichnisses angezeigt.

Speichern in

Wählen Sie das Verzeichnis aus, dessen Inhalt Sie zu sehen wünschen. Im Listenfeld Dateien erscheinen die Dateien des aktuellen Verzeichnisses, die zu den Platzhalterzeichen im Eingabefeld Dateiname oder zum Dateityp im Kombinationsfeld Dateityp passen.

Laufwerke

Wählen Sie das aktuelle Laufwerk. Die Verzeichnisstruktur des aktuellen Laufwerks erscheint im Listenfeld Speichern in.

Siehe auch

Bild-Editor (siehe Seite 640)

2.290 String-Liste speichern (Dialogfeld)

Verwenden Sie das Dialogfeld String-Liste speichern, um die String-Liste aus dem String-Listen-Editor in einer Textdatei zu speichern.

So öffnen Sie dieses Dialogfeld:

1. Öffnen Sie den String-Listen-Editor.
2. Klicken Sie mit der rechten Maustaste und wählen Sie Speichern.

Optionen des Dialogfeldes String-Liste speichern

Folgende Optionen stehen im Dialogfeld String-Liste speichern zur Verfügung:

Dateiname

Geben Sie den Namen der zu speichernden Datei oder die im Listenfeld Dateien als Filter zu verwendenden Platzhalterzeichen ein.

Dateien (großes Listenfeld)

Im Listenfeld Dateien werden die Dateien des aktuellen Verzeichnisses angezeigt, die zu den Platzhalterzeichen im Eingabefeld Dateiname oder zum Dateityp im Kombinationsfeld Dateityp passen.

Dateityp

Wählen Sie einen Filter zur Anzeige der verschiedenen Dateitypen. Standardmäßig werden im Listenfeld Dateien die Textdateien (*.txt) im aktuellen Verzeichnis angezeigt.

Speichern in

Wählen Sie das Verzeichnis aus, dessen Inhalt Sie zu sehen wünschen. Im Listenfeld Dateien erscheinen die Dateien des aktuellen Verzeichnisses, die zu den Platzhalterzeichen im Eingabefeld Dateiname oder zum Dateityp im Kombinationsfeld Dateityp passen.

Laufwerke

Wählen Sie das aktuelle Laufwerk. Die Verzeichnisstruktur des aktuellen Laufwerks erscheint im Listenfeld Speichern in.

Siehe auch

String-Listen-Editor (siehe Seite 625)

2.291 Menü auswählen

Kontextmenü des Menü-Designers►Menü auswählen

Verwenden Sie dieses Dialogfeld, um aus einer Liste der dem Formular zugeordneten Menüs ein Menü auszuwählen, dass im Menü-Designer geöffnet werden soll.

2.292 Parameter ändern

Refactor ► Parameter ändern

Im Dialogfeld Parameter ändern können Sie die Parameter einer Methode einfügen, bearbeiten, entfernen oder neu anordnen.

Element	Beschreibung
Klasse	Zeigt die Klasse an, in der die ausgewählte Methode definiert ist.
Methode	Zeigt die Methode an, die mit dem Refactoring bearbeitet wird.
Parameter	Führt die Informationen über die Parameter in einer Liste auf, die in der Methode deklariert sind.
Hinzufügen	Zeigt das Dialogfeld Parameter hinzufügen an, das Sie dazu verwenden können, einen Parameter zur Methodensignatur hinzuzufügen.
Bearbeiten	Zeigt das Dialogfeld Parameter bearbeiten an, das Sie dazu verwenden können, einen bereits erstellten Parameter zu bearbeiten.
Entfernen	Der ausgewählte String wird gelöscht.
Nach oben	Verschiebt den ausgewählten Parameter in der Liste nach oben.
Nach unten	Verschiebt den ausgewählten Parameter in der Liste nach unten.

Anmerkung: Wenn Sie einen Parameter entfernen, müssen Sie jede Methode manuell entfernen, die den gelöschten Parameter verwendet.

Siehe auch

Parameter ändern (siehe Seite 1422)

Parametertypen (siehe Seite 938)

Grundlegende syntaktische Elemente (siehe Seite 857)

2.293 Parameter hinzufügen

Verwenden Sie das Dialogfeld Parameter hinzufügen oder Parameter bearbeiten dazu, einer Methodensignatur einen Parameter hinzuzufügen oder einen bereits vorhandenen zu bearbeiten.

Element	Beschreibung
Parametername	Legt den Namen des Parameters fest. Sie müssen einen zulässigen Delphi-Bezeichner eingeben.
Datentyp	Legt den Typ des neuen Parameters fest.
Übereinstimmende Ergebnisse	Führt jene Datentypen auf, die dem Text entsprechen, der in das Feld Datentyp eingegeben haben.
Literaler Wert	Gibt den Wert für den neuen Parameter an. Das Refactoring verwendet dieses Literal in vorhandenen Aufrufen dieser Prozedur als Wert des neuen Parameters.
Parametertyp	Gibt den Parametertyp an.
Wert	Übergibt den Parameter anhand des Wertes.
Var	Übergibt den Parameter anhand der Referenz.
Out	Übergibt den Parameter anhand der Referenz und löscht den ursprünglichen Wert.
Const	Übergibt den Parameter anhand des Wertes und unterbindet die Zuweisung an den Parameter.

Siehe auch

Parameter ändern ([siehe Seite 1422](#))

Parametertypen ([siehe Seite 938](#))

Grundlegende syntaktische Elemente ([siehe Seite 857](#))

2.294 Namespace hinzufügen

Refactor ► Namespace importieren

Verwenden Sie dieses Dialogfeld, um Namespaces in die **uses**-Klausel der Quelltextdatei zu importieren. Dies wird auf der Basis von Objekten im Quelltext ausgeführt, die in diesem Namespace enthalten sind, aber für die noch kein Namespace deklariert wurde.

Element	Beschreibung
Suchen	Enthält per Vorgabe das ausgewählte Objekt. Sie können einen anderen Objektnamen eingeben, um eine vollständig neue Liste mit Namespaces anzuzeigen. Wenn dieses Feld leer ist, werden in der Liste Übereinstimmende Ergebnisse alle verfügbaren Namespaces aufgeführt.
Übereinstimmende Ergebnisse	Zeigt eine Liste der für das angegebene Objekt passenden Namespaces an. Der Namespace, der am wahrscheinlichsten das Objekt enthält, ist markiert. Sie aber auch einen anderen oder mehrere andere Namespaces aus der Liste auswählen.

2.295 Felder deklarieren

Refactor ▶ Feld deklarieren

Verwenden Sie dieses Dialogfeld, um in Ihrem Quelltext ein Feld zu deklarieren.

Element	Beschreibung
Aktuelle Klasse	Zeigt die Klasse an, von der das Feld abgeleitet wird.
Feldname	Zeigt den Namen des Feldes an, das deklariert werden soll. Standardmäßig wird der Name angezeigt, den Sie im Quelltext-Editor eingegeben haben. Wenn Sie in dieses Feld einen ungültigen Namen eingeben, wie z.B. ein reserviertes Wort, wird im Dialogfeld ein  angezeigt, das Sie darauf hinweist, den Fehler zu korrigieren.
Typ	Legt den Datentyp des Feldes fest.
Array	Legt fest, dass das Feld in einem Array deklariert werden soll.
Dimensionen	Legt die Dimensionen für das Array fest.
Sichtbarkeit	Legt die Sichtbarkeit des Feldes fest. Wenn der Wert in Feldname mit einem vorhandenen Feldnamen im selben Gültigkeitsbereich in Konflikt steht, wird das Dialogfeld Refactoring geöffnet, in dem der Konflikt angezeigt wird.

Siehe auch

Überblick zum Refactoring (siehe Seite 1412)

2.296 Variablen deklarieren

Refactor>Variable deklarieren

Verwenden Sie dieses Dialogfeld, um eine lokale Variable in einer Prozedur zu deklarieren. Wenn sich der Cursor im Quelltext-Editor nicht auf einer undeklärten Variable befindet, steht der Befehl **Refactor>Variable deklarieren** nicht zur Verfügung.

Element	Beschreibung
Name	Zeigt den Namen der aktuell ausgewählten Variable an.
Typ	Legt den Typ der Variable fest. Standardmäßig wird der Name angezeigt, den Sie im Quelltext-Editor eingegeben haben. Wenn Sie in dieses Feld einen ungültigen Namen eingeben, wie z.B. ein reserviertes Wort, wird im Dialogfeld ein  angezeigt, das Sie darauf hinweist, den Fehler zu korrigieren.
Array	Legt fest, dass die Variable in einem Array deklariert werden soll.
Dimensionen	Legt die Dimensionen für das Array fest.
Wert	Initialisiert die Variable mit dem angegebenen Wert.

Siehe auch

Überblick zum Refactoring (siehe Seite 1412)

2.297 Methode extrahieren

Refactor ▶ Methode extrahieren

In diesem Dialogfeld können beispielsweise ein markiertes Codefragment in eine Methode konvertieren. RAD Studio verschiebt den extrahierten Code an eine Position außerhalb der aktuellen Methode, legt die erforderlichen Parameter fest, erzeugt nach Bedarf lokale Variablen, bestimmt den Rückgabetyp und ersetzt das ursprüngliche Codefragment durch einen Aufruf der neuen Methode.

Element	Beschreibung
Aktuelle Methode	Zeigt den Namen der Methode an, in der sich das Codefragment aktuell befindet.
Neue Methode	Legt den Namen für die neu extrahierte Methode fest. ExtractedMethod wird als Vorgabe angezeigt, was Sie aber überschreiben können.
Extrahierter Beispielcode	Zeigt den Code an, der für die neu extrahierte Methode erzeugt wird.

Siehe auch

Überblick zum Refactoring (siehe Seite 1412)

2.298 Ressourcen-String extrahieren

Refactor ▶ Ressourcen-String extrahieren

Verwenden Sie dieses Dialogfeld, um den aktuell im Quelltext-Editor markierten String in einen Ressourcen-String zu konvertieren. Das Schlüsselwort `resourcestring` und der Ressourcen-String werden dem Abschnitt `implementation` des Quelltextes hinzugefügt und der Original-String wird durch neuen Ressourcen-String-Namen ersetzt.

Element	Beschreibung
String	Zeigt den Namen des Strings an, der als Ressourcen-String extrahiert werden soll.
Name	Zeigt einen Vorschlag für den String-Namen an. Sie können diesen vorgeschlagenen Namen ändern.

Siehe auch

Überblick zum Refactoring (siehe Seite 1412)

[Ressourcen auslagern](#)

2.299 Unit suchen

Refactor ► Unit suchen

Verwenden Sie dieses Dialogfeld, um Units zu suchen und sie der **uses**-Klausel der Delphi-Quelltextdatei hinzuzufügen.

Element	Beschreibung
Suchen	Zeigt den ausgewählten Such-Bezeichner an. In diesem Textfeld können Sie den Bezeichner ändern. Die Suche wird während der Eingabe automatisch eingegrenzt.
Übereinstimmende Ergebnisse	Zeigt alle potentiell passenden Units an. Wählen Sie eine oder mehrere Units aus der Liste aus, die als Unit-Referenzen der uses -Klausel hinzugefügt werden sollen.
Hinzufügen Interface	zu: Markieren Sie dieses Feld, um die Unit-Referenz dem interface -Abschnitt hinzuzufügen.
Hinzufügen Implementation	zu: Markieren Sie dieses Feld, um die Unit-Referenz dem implementation -Abschnitt hinzuzufügen.

2.300 Refactorings

Ansicht ▾ Refactorings

Verwenden Sie dieses Dialogfeld, um die aufgeführten Refactorings durchzuführen.

Element	Beschreibung
Refactor	Führt die Refactoring-Operation für die Einträge durch.
Refactoring rückgängig machen	Macht alle Refactorings rückgängig, die unmittelbar vor dem Aufruf dieses Befehls durchgeführt wurden.
Refactoring entfernen	Entfernt die ausgewählten Refactorings aus der Liste.
Alle Refactorings entfernen	Entfernt alle Refactorings aus der Liste.

Siehe auch

Überblick zum Refactoring (siehe Seite 1412)

2.301 Symbol umbenennen (C++)

Refactor ▶ Umbenennen

Verwenden Sie dieses Dialogfeld dazu, einen neuen Namen für das ausgewählte Symbol anzugeben, ehe Sie den Code mit einer Refactoring-Operation bearbeiten.

Element	Beschreibung
Alter Name	Zeigt den aktuellen Symbolnamen an.
Neuer Name	Geben Sie den neuen Namen für die Symboldatei ein.
Vor Refactoring Verweise anzeigen	Zeigt das Dialogfeld Refactorings an, in dem Sie die vorgeschlagenen Änderungen überprüfen und das Symbol selektiv umbenennen können. Wenn diese Option nicht markiert ist, wird die Refactoring-Umbenennung sofort ausgeführt.

2.302 Umbenennen (C#)

Refactor ▶ Umbenennen

Verwenden Sie dieses Dialogfeld, um mit dem aktuell im Quelltext-Editor markierten Symbol, wie z.B. eine Variable, einen Typ, eine Methode oder einen Parameter, eine Refactoring-Umbenennung durchzuführen. Das erste Feld in diesem Dialogfeld ist abhängig vom Typ des Symbols, das umbenannt werden soll.

Element	Beschreibung
Namespace	Wird beim Umbenennen eines Typs angezeigt. Gibt den Namespace an, in dem der Typ definiert ist.
Prozedur	Wird beim Umbenennen einer Variablen angezeigt. Gibt die Prozedur an, in der die Variable ausgewählt ist.
Class	Wird beim Umbenennen eines Felds, einer Methode oder eines Parameters angezeigt. Gibt die Klasse an, in der das Feld, die Methode oder der Parameter definiert ist.
Alter Name	Zeigt den aktuellen Namen des markierten Symbols an.
Neuer Name	Geben Sie den neuen Namen für das markierte Symbol ein.
Vor Refactoring Verweise anzeigen	Zeigt das Dialogfeld Refactorings an, in dem Sie die vorgeschlagenen Änderungen überprüfen und das Symbol selektiv umbenennen können. Wenn diese Option nicht markiert ist, wird die Refactoring-Umbenennung sofort ausgeführt.

Tip: Der Befehl **Refactor ▶ Umbenennen** steht auch im Kontextmenü des Quelltext-Editors zur Verfügung.

Siehe auch

Überblick zum Refactoring (siehe Seite 1412)

2.303 Umbenennen (Delphi)

Refactor ▶ Umbenennen

Verwenden Sie dieses Dialogfeld, um mit dem aktuell im Quelltext-Editor markierten Symbol, wie z.B. eine Variable, einen Typ, eine Methode oder einen Parameter, eine Refactoring-Umbenennung durchzuführen. Das erste Feld in diesem Dialogfeld ist abhängig vom Typ des Symbols, das umbenannt werden soll.

Element	Beschreibung
Unit	Wird beim Umbenennen eines Typs angezeigt. Gibt die Unit an, in der der Typ definiert ist.
Prozedur	Wird beim Umbenennen einer Variablen angezeigt. Gibt die Prozedur an, in der die Variable ausgewählt ist.
Klasse	Wird beim Umbenennen eines Felds, einer Methode oder eines Parameters angezeigt. Gibt die Klasse an, in der das Feld, die Methode oder der Parameter definiert ist.
Alter Name	Zeigt den aktuellen Namen des markierten Symbols an.
Neuer Name	Geben Sie den neuen Namen für das markierte Symbol ein.
Vor Refactoring Verweise anzeigen	Zeigt das Dialogfeld Refactorings an, in dem Sie die vorgeschlagenen Änderungen überprüfen und das Symbol selektiv umbenennen können. Wenn diese Option nicht markiert ist, wird die Refactoring-Umbenennung sofort ausgeführt.

Tip: Der Befehl **Refactor ▶ Umbenennen** steht auch im Kontextmenü des Quelltext-Editors zur Verfügung.

Siehe auch

Überblick zum Refactoring (siehe Seite 1412)

2.304 Mit Prozess verbinden

Start ▶ Mit Prozess verbinden

Verwenden Sie dieses Dialogfeld zum Debuggen eines Prozesses, der aktuell auf einem lokalen oder externen Computer ausgeführt wird.

Element	Beschreibung
Debugger	Wählen Sie den geeigneten Debugger aus der Liste mit den registrierten Debuggern aus. Wenn Sie den Borland .NET-Debugger auswählen, werden nur die verwalteten Prozesse in der Liste Ausgeführte Prozesse angezeigt. Wenn Sie den Borland Win32-Debugger auswählen, werden sowohl verwaltete als auch nicht verwaltete Prozesse angezeigt.
Externer Rechner	Der Name des externen Geräts, die jene Anwendung ausführt, die vom Debugger benutzt werden soll.
Laufende Prozesse	Führt die aktuell auf dem lokalen oder falls angegeben, auf dem externen Computer ausgeführten Prozesse auf. Beachten Sie, dass der externe Server ausgeführt werden muss.
PID	Führt die Prozessbezeichner des Prozesses auf.
Path	Enthält die Position des Prozesses.
Systemprozesse anzeigen	Wird eingeblendet, wenn bei der Option Debugger der Borland Win32-Debugger ausgewählt ist. Schließt die Systemprozesse in die Liste Ausgeführte Prozesse ein.
Pause nach Verbindung	Hält den Prozess an, nachdem der Debugger mit dem Prozess verbunden wurde und zeigt das CPU-Fenster an. Damit die Ausführung fortgesetzt wird, müssen Sie einen der Befehle Start, Gesamte Routine oder Einzelne Anweisung aufrufen.
Aktualisieren	Aktualisiert die Liste der ausgeführten Prozesse und zeigt sie erneut an.
Verbinden	Verbindet den markierten Prozess mit dem Debugger und zeigt, wenn die Option Pause nach Verbindung aktiviert ist, das CPU-Fenster an. Die Schaltfläche Verbinden ist für die IDE und alle Prozesse, die bereits mit dem Debugger verbunden sind, deaktiviert.

2.305 Änderung

Verwenden Sie dieses Dialogfeld, um dem aktuell auf der Registerkarte Daten des Debug-Inspektors ausgewählten Datenelement einen neuen Wert zuzuweisen.

2.306 Benachrichtigung über Debugger-Exception

Dieses Dialogfeld wird angezeigt, wenn das Programm, das Sie gerade testen, eine Sprach- oder Betriebssystem-Exception auslöst und Sie für den Debugger auf den Seiten Sprach-Exceptions und Native BS-Exceptions des Dialogfeldes **Tools>Optionen>Debugger-Optionen** Exceptions-Behandlungsoptionen festgelegt haben.

Das Format der Benachrichtigung ist:

Im Projekt <Projektname> ist eine Exception der Klasse <yyyy> aufgetreten. Meldung: <Meldungstext>.

Wenn die Zeichen *yyyy* in der Meldung einen Klassennamen darstellen, handelt es sich bei der Exception um eine Sprach-Exception. Wenn *yyyy* ein hexadezimaler Wert ist, handelt es sich um eine Betriebssystem-Exception.

Element	Beschreibung
Diesen Exception-Typ ignorieren	Veranlasst den Debugger, diesen Sprach-Exception- oder BS-Exception-Typ zu ignorieren und markiert das zugehörige Kontrollkästchen in der Liste Folgende Exception-Typen ignorieren auf der Seite Tools>Optionen>Debugger-Optionen>Sprach-Exceptions . Wenn BS-Exceptions ignoriert werden sollen, wird die Option Behandelt von auf der Seite Native BS-Exceptions für alle Exception-Bereiche auf Debugger gesetzt, in die die ausgelöste Exception fällt.
Exception-Objekt untersuchen	Zeigt nur Exceptions an, wenn der Borland .NET Debugger verwendet wird. Zeigt für eine aufgetretene Exception das Dialogfeld Debug-Inspektor an, wenn Sie auf Anhalten klicken, um die Ausführung zu stoppen. Wenn Sie auf Fortsetzen klicken, hat diese Option keine Auswirkungen.
CPU-Ansicht anzeigen	Diese Option wird nur angezeigt, wenn die Position der Exception keiner Position im Quelltext entspricht. Zeigt das CPU-Fenster an, wenn Sie auf Anhalten klicken, um die Ausführung zu stoppen. Wenn das Kontrollkästchen CPU-Ansicht anzeigen eingeblendet ist und Sie es nicht markiert haben, durchsucht die IDE den Aufruf-Stack nach einem Aufruf in dem Stack, der Quellcode enthält und zeigt den ersten gefundenen Aufruf an. Wenn Sie auf Fortsetzen klicken, hat diese Option keine Auswirkungen.
Anhalten	Hält die Programmausführung an der Position an, an der die Exception auftrat und blendet die Codezeile im Quelltext-Editor ein.
Fortsetzen	Setzt die Programmausführung fort.

Tip: Um diese oder ähnliche Meldungen in die Zwischenablage zu kopieren, drücken Sie STRG+C.

Anmerkung: In manchen Fällen verhindert der Programmstatus die weitere Ausführung und das Dialogfeld Benachrichtigung über Debugger-Exception wird wiederholt angezeigt. In diesem Fall müssen Sie Anhalten anklicken und dann **Start>Programm zurücksetzen** wählen, um die aktuelle Programmausführung zu beenden und den Arbeitsspeicher freizugeben.

2.307 Debug-Inspektor

Start ▶ Untersuchen

Verwenden Sie dieses Dialogfeld, um die folgenden Datentypen zu überprüfen: Arrays, Klassen, Konstanten, Funktionen, Zeiger, skalare Variablen und Schnittstellen.

Der Debug-Inspektor enthält drei Bereiche:

Bereich	Beschreibung
Oberer Bereich	Zeigt den Namen, den Typ und die Adresse oder die Arbeitsspeicherposition des untersuchten Elements an. Beim Untersuchen eines Funktionsaufrufs, der ein Objekt, einen Record oder ein Array zurückgibt, zeigt der Debugger "Im Debugger" anstelle der temporär zugewiesenen Adresse an.
Mittlerer Bereich	Zeigt abhängig vom Typ der untersuchten Daten ein oder mehrere der folgenden Register an: Register Daten - Zeigt Datennamen (oder Klassendatenelemente) und aktuelle Werte an. Register Methoden - Wird nur angezeigt, wenn eine Klasse oder ein Interface untersucht wird, und zeigt Klassenmethoden (Elementfunktionen) und aktuelle Adresspositionen an. Register Eigenschaften - Wird nur angezeigt, wenn eine Objektklasse mit Eigenschaften untersucht wird und zeigt Eigenschaftsnamen und aktuelle Werte an.
Unterer Bereich	Zeigt den Datentyp des aktuell im mittleren Bereich markierten Elements an.
Statuszeile	Zeigt den Datentyp des untersuchten Elements an.

Kontextmenü

Klicken Sie im Debug-Inspektor mit der rechten Maustaste, um die folgenden Befehle anzuzeigen.

Element	Beschreibung
Änderung	Ermöglicht es Ihnen, einem Datenelement einen neuen Wert zuzuweisen. Mehrere Punkte (...) erscheinen neben einem Element, das geändert werden kann. Diese Schaltfläche kann anstatt des Befehls Ändern angeklickt werden. Dieser Befehl ist nur aktiviert, wenn die untersuchten Daten geändert werden können.
Vererbung anzeigen	Schaltet die Ansicht in den Ausschnitten Daten, Methoden und Eigenschaften zwischen zwei Modi um: einem, der alle intrinsischen und vererbten Datenelemente oder Eigenschaften einer Klasse anzeigt und einem, der nur die in der Klasse deklarierten Elemente anzeigt.
Voll qualifizierte Namen anzeigen	Diese Option ermöglicht die Anzeige der geerbten Elemente mit ihrem voll qualifizierten Namen.
Sortierung	Sortiert die Datenelemente, die im Debug-Inspektor angezeigt werden, nach Namen oder in der Reihenfolge, in der sie im Quelltext deklariert sind
Untersuchen	Öffnet für das markierte Datenelement ein neues Debug-Inspektor-Fenster. Dies ist hilfreich bei der Überprüfung der Einzelheiten von Datenstrukturen Klassen und Arrays.
Absteigend	Entspricht dem Befehl Untersuchen. Das aktuelle Debug-Inspektor-Fenster wird allerdings ersetzt durch die gerade untersuchten Details (ein neues Debug-Inspektor-Fenster wird nicht geöffnet). Um zu einer höheren Ebene zurückzukehren, verwenden Sie die Aufzeichnungsliste.
Neuer Ausdruck	Ermöglicht Ihnen, einen neuen Ausdruck zu untersuchen.
Typumwandlung	Ermöglicht es Ihnen, einen anderen Datentyp für ein Element anzugeben, das Sie inspizieren möchten. Typumwandlung ist hilfreich, wenn der Debug-Inspektor ein Symbol enthält, für das keine Typinformationen zur Verfügung stehen und wenn Sie explizit den Typ für untypisierte Zeiger setzen wollen.

Andockbar	Legt fest, ob das Fenster Debug-Inspektor angedockt werden kann.
Immer im Vordergrund	Belässt das Fenster sichtbar, wenn es den Fokus verliert.

2.308 Auswerten/Ändern

Start ▶ Auswerten/Ändern

Verwenden Sie dieses Dialogfeld zum Auswerten oder Ändern des Wertes eines vorhandenen Ausdrucks oder einer Eigenschaft. Dies ist nützlich, wenn Sie eine Quelltextkorrektur testen, ohne den Debugger zu beenden, den Quelltext ändern und das Programm erneut compilieren möchten.

Element	Beschreibung
Auswerten	Wertet den Ausdruck im Eingabefeld Ausdruck aus und zeigt den Wert im Eingabefeld Ergebnis an.
Ändern	Ändert den Wert des Ausdrucks im Eingabefeld Ausdruck und zeigt den Wert im Eingabefeld Neuer Wert an.
Überwachen	Erzeugt einen überwachten Ausdruck für den ausgewählten Ausdruck.
Untersuchen	Öffnet für das markierte Datenelement ein neues Debug-Inspektor-Fenster. Dies ist hilfreich bei der Überprüfung der Einzelheiten von Datenstrukturen Klassen und Arrays.
Ausdruck	Geben Sie die Variable, das Feld, das Array oder das Objekt ein, das ausgewertet oder geändert werden soll. Standardmäßig wird das Wort an der Cursorposition im Quelltext-Editor in das Eingabefeld Ausdruck übernommen. Sie können diesen Ausdruck übernehmen, einen anderen eingeben oder einen Ausdruck aus der Liste der bereits ausgewerteten Ausdrücke auswählen. Zum Auswerten einer Funktion geben Sie den Funktionsnamen, Klammern und Argumente genauso ein, wie Sie sie in Ihr Programm eingeben würden; lassen Sie aber das Anweisungsende-Semikolon (;) weg.
Ergebnis	Zeigt den Wert des im Textfeld Ausdruck angegebenen Elements nach dem Aufruf von Auswerten oder Ändern an.
Neuer Wert	Weist dem im Eingabefeld Ausdruck angegebenen Element einen neuen Wert zu. Geben Sie einen neuen Wert für das Element ein, wenn Sie dessen Wert ändern möchten.

Anmerkung: Sie können jeden gültigen Ausdruck oder statische Variablen auswerten, die sich aktuell nicht im Gültigkeitsbereich der Anwendung befinden.

Formatbezeichner für die Anzeige

Per Vorgabe zeigt der Debugger das Ergebnis in dem Format an, das dem Datentyp des Ausdrucks entspricht. Beispielsweise werden Integerwerte im Dezimalformat angezeigt. Um das Anzeigeformat zu ändern, geben Sie ein Komma (,) und daran anschließend einen Formatbezeichner hinter dem Ausdruck ein.

Folgende Tabelle beschreibt die Formatbezeichner für Auswerten/Ändern.

Bezeichner	Betroffene Typen	Beschreibung
,C	Char, Strings	Zeichen. Gibt die Zeichen für die ASCII-Werte 0 bis 31 in der #nn-Notation von Delphi aus.
,S	Char, Strings	String. Gibt die ASCII-Werte 0 bis 31 in der #nn-Notation von Delphi aus.
,D	Integers	Dezimal. Zeigt Integer als Dezimalzahlen an. Gilt auch für Integer-Werte in Datenstrukturen.
,H oder ,X	Integers	Hexadezimal. Zeigt Integer-Werte hexadezimal mit Präfix \$ an, einschließlich derer in Datenstrukturen.

,Fn	Fließkomma	Gleitkomma. Zeigt <i>n</i> signifikante Ziffern an, wobei <i>n</i> von 2 bis 18 gehen kann. Um beispielsweise die ersten vier Ziffern eines Gleitkommawertes anzuzeigen, geben Sie ,F4 ein. Ist <i>n</i> nicht angegeben, wird als Standardwert 11 verwendet.
-----	------------	---

2.309 Package-Import suchen

Start▶Start (F9)

Das Dialogfeld erscheint, wenn Ihre Anwendung ein in unter **Projekt▶Optionen▶Packages** angegebenes Laufzeit-Package nicht finden kann.

Element	Beschreibung
Package-Import	Enthält den Namen des Package, das nicht gefunden wurde.
Durchsuchen	Zeigt ein Dialogfeld zum Suchen von Dateien an, in dem Sie den korrekten Package-Namen auswählen können.
Diese Referenz entfernen	Löscht den Namen des Package-Import aus der Liste der Laufzeit-Packages des Projekts.
Nicht wieder fragen	Legt fest, dass RAD Studio mit dem Laden der Laufzeit-Packages fortfahren soll, ohne dieses Dialogfeld erneut anzuzeigen.

Siehe auch

Überblick über Packages (siehe Seite 1002)

Laufzeit-Packages

Packages in Anwendungen laden

2.310 Untersuchen

Start▶Untersuchen

Verwenden Sie dieses Dialogfeld zum Eingeben des Ausdruckes, den Sie im Debug-Inspektor untersuchen möchten.

Element	Beschreibung
Ausdruck	Geben Sie einen gültigen Ausdruck ein.

Siehe auch

Werte von Datenelementen untersuchen und ändern ( siehe Seite 24)

2.311 Prozess laden Umgebungsblock

[Start](#)▶[Prozess laden](#)▶[Umgebungsblock](#)

Verwenden Sie diese Seite zur Angabe der Umgebungsvariablen, die während des Debuggens an die Anwendung übergeben werden.

Element	Beschreibung
Debugger	Gibt den Namen des verwendeten Debuggers an. Sie können je nach Art der Anwendung den Borland Win32-Debugger oder den Borland .NET Debugger verwenden.
Systemvariablen	Zeigt eine Liste aller Umgebungsvariablen und ihrer Werte an, die auf Systemebene definiert sind. Eine Systemvariable kann nicht gelöscht, aber überschrieben werden.
Variable überschreiben	Öffnet das Dialogfeld Variable überschreiben, in dem Systemvariablen geändert werden können, um neue vom Anwender überschriebene Variablen zu erstellen. Diese Schaltfläche ist deaktiviert (abgedunkelt), bis Sie eine Variable in der Liste Systemvariablen auswählen.
Vom Anwender überschrieben	Zeigt alle Anwendervariablen mit den zugehörigen Werten an. Anwendervariablen haben Vorrang vor Systemvariablen und verlieren ihre Gültigkeit erst, wenn sie gelöscht werden.
Neu	Öffnet das Dialogfeld Neue Anwendervariable, in dem Sie eine neue Anwendervariable aus einer Systemvariable erstellen können.
Bearbeiten	Öffnet das Dialogfeld Anwendervariable bearbeiten, in dem Sie die Anwendervariable ändern können, die in der Liste Vom Anwender überschrieben ausgewählt ist.
Löschen	Löscht die Anwendervariable, die in der Liste Vom Anwender überschrieben ausgewählt ist.
Mit Systemvariablen	Übergibt die Systemumgebungsvariablen an die Anwendung, für die der Debug-Vorgang ausgeführt wird. Wenn diese Option deaktiviert ist, werden nur die vom Anwender überschriebenen Variablen an die Anwendung übergeben.

2.312 Prozess laden Lokal

Start▶Prozess laden▶Lokal

Verwenden Sie dieses Dialogfeld zur Übergabe von Befehlszeilenparametern an die Anwendung, zur Festlegung einer Host-Anwendung für den Test einer DLL oder zum Laden einer ausführbaren Datei in den Debugger.

Element	Beschreibung
Debugger	Gibt den Namen des verwendeten Debuggers an. Sie können entweder den Borland Win32 Debugger oder den Borland .NET Debugger aus dem Pulldown-Menü laden.
Prozess	Geben Sie den Pfad zu der EXE-Datei an, die im Debugger ausgeführt werden soll. Klicken Sie dann auf Laden, um die betreffende Datei zu laden. Die Ausführung dieser Datei wird am Einsprungpunkt angehalten. Wenn dort keine Debug-Informationen zur Verfügung stehen, wird das Fenster CPU geöffnet. Wählen Sie Start▶Start , um die ausführbare Datei auszuführen.
Parameter	Geben Sie hier die Kommandozeilenargumente ein, die beim Starten an Ihre Anwendung übergeben werden sollen.
Arbeitsverzeichnis	Geben Sie hier das Debug-Verzeichnis an. Standardmäßig wird das Verzeichnis mit der ausführbaren Datei der Anwendung verwendet.
Start-Code beim Laden ausführen	Führt den Start-Code aus, der beim Erstellen des Projekts automatisch generiert wurde. Dieser Code wird vor dem Erreichen des Haupteinsprungpunkts des Programms ausgeführt.
Laden	Lädt die Anwendung (der Prozess wird geladen und angehalten).

2.313 Prozess laden Extern

Start▶**Prozess laden**▶**Extern**

Verwenden Sie dieses Dialogfeld, um eine Verbindung zu einem externen Computer herzustellen, auf dem der Remote-Debug-Server ausgeführt wird, und eine Sitzung mit dem Remote-Debugger zu starten.

Element	Beschreibung
Debugger	Gibt den Namen des verwendeten Debuggers an. Sie können je nach Art der Anwendung den CodeGear Win32-Debugger oder den CodeGear .NET Debugger verwenden.
Externer Pfad	Geben Sie den Pfad der .exe-Datei auf dem externen Computer relativ zu dem Verzeichnis ein, in dem sich der Remote-Debug-Server (rmtdbg105.exe) befindet.
Externer Host	Geben Sie den Namen oder die TCP/IP-Adresse des externen Computers an, auf dem die Anwendung ausgeführt werden soll. Der Remote-Debug-Server (rmtdbg100.exe) muss auf dem externen Computer ausgeführt werden. Wenn beim Start von rmtdbg100.exe ein Port angegeben wurde, geben Sie nach dem Host-Namen einen Doppelpunkt ein, gefolgt vom Port. Angenommen, Sie haben Port 8000 angegeben, geben Sie den externen Hosts wie folgt an hostname:8000 oder 127.0.0.1:8000 . Andernfalls wird der Standard-Port 64447 verwendet.
Parameter	Geben Sie hier die Befehlszeilenargumente ein, die beim Starten an Ihre Anwendung bzw. an die Host-Anwendung übergeben werden sollen.
Arbeitsverzeichnis	Geben Sie hier das Debug-Verzeichnis an. Standardmäßig wird das Verzeichnis mit der ausführbaren Datei der Anwendung verwendet.
Start-Code beim Laden ausführen	Führt den Start-Code aus, der beim Erstellen des Projekts automatisch generiert wurde. Dieser Code wird vor dem Erreichen des Haupteinsprungspunkts des Programms ausgeführt.

Siehe auch

Debuggen externer Anwendungen (siehe Seite 32)

2.314 Prozess laden Symboltabellen

[Start](#) [Prozess laden](#) [Symboltabellen](#)

Verwenden Sie dieses Dialogfeld, um den Speicherort der Symboltabellen anzugeben, die bei der Fehlersuche verwendet werden sollen.

Element	Beschreibung
Debugger	Gibt den Namen des verwendeten Debuggers an. Sie können je nach Art der Anwendung den Borland Win32-Debugger oder den Borland .NET Debugger verwenden.
Suchpfad Debug-Symbole	für Gibt das Verzeichnis an, das die Symboltabellen enthält, die bei der Fehlersuche verwendet werden sollen. Wenn das Kontrollfeld Alle Symbole laden aktiv ist, wird dieser Pfad verwendet.
Alle Symbole laden	Definiert den Status der Liste für die Zuordnungen des Modulnamens zum Symboltabellenpfad. Ist das Kontrollfeld aktiviert, ist die Auswahlliste deaktiviert und alle Symboltabellen werden vom Debugger geladen. Der Debugger verwendet den Suchpfad für Debug-Symbole, um nach der jeweilis zu dem Modul gehörenden Symboltabellendatei zu suchen, das von dem aktuell zu überprüfenden Prozess geladen wird. Ist das Kontrollfeld nicht aktiviert, ist die Liste für die Zuordnungen des Modulnamens zum Symboltabellenpfad aktiv und die dortigen Einstellungen werden benutzt.
Zuordnungen Modulnamens des zum Symboltabellenpfad	Zeigt die aktuelle Zuordnung jedes Modulnamens zum Suchpfad einer Symboltabelle an, die für das Projekt definiert ist. Mit den nach oben bzw. nach unten weisenden Pfeilen rechts neben der Liste mit den Suchpfaden können Sie einen markierten Pfad in der Suchreihenfolge nach oben bzw. nach unten verlagern. Der Debugger durchsucht diese Liste, um einen entsprechenden Namen des zu ladenden Moduls zu finden. Hat der Debugger einen entsprechenden Modulnamen gefunden, verwendet er den zugehörigen Pfad, um die Symboltabelle dieses Moduls zu suchen. Angenommen, das Modul foo123.dll ist geladen und die Liste zeigt foo*.dll als erstes Element und *123.dll als ein nachfolgendes Element an, so verwendet der Debugger nur den Symboltabellenpfad für foo*.dll , auch wenn beide Elemente dem Modul entsprechen, das geladen wurde.
Symbolen unspezifizierte für Module laden	Gibt an, ob Symboltabellen für Module, die nicht in der Liste der Zuordnungen des Modulnamens zum Symboltabellenpfad enthalten sind (entweder explizit oder über eine Dateimaske), bei der Fehlersuche geladen werden. Ist das Kontrollfeld aktiv, werden die Symboltabellen für nicht definierte Module über den Suchpfad für Debug-Symbole geladen. Ist das Kontrollfeld nicht aktiv, werden die Symboltabellen nur für die Module in der Liste geladen.
Neu	Zeigt das Dialogfeld Suchpfad für Symboltabelle hinzufügen an, in dem Sie einen Modulnamen und einen zugehörigen Suchpfad für Tabellen angeben können. Das Modul und der Pfad werden der Liste der Zuordnungen des Modulnamens zum Symboltabellenpfad hinzugefügt. Beachten Sie, dass Sie einen leeren Pfad hinzufügen können, um zu verhindern, dass eine Symboltabelle für ein Modul geladen wird.
Bearbeiten	Zeigt das ausgewählte Modul und den Pfad im Dialogfeld Suchpfad für Symboltabelle hinzufügen an, wodurch Sie den Modulnamen oder Pfad bearbeiten können, der in der Liste Mappings from Module Name to Symbol Table Path angezeigt wird.
Löschen	Entfernt das ausgewählte Modul aus der Liste der Zuordnungen des Modulnamens zum Symboltabellenpfad.

Siehe auch

Anwendungen debuggen ([siehe Seite 1439](#))

Dateien für das externe Debuggen vorbereiten ([siehe Seite 35](#))

Lokal ([siehe Seite 666](#))

Extern ([siehe Seite 667](#))

2.315 Neuer Ausdruck

Verwenden Sie dieses Dialogfeld, um einen neuen Ausdruck zu untersuchen. Geben Sie den Ausdruck ein oder wählen Sie einen zuvor eingegebenen aus der Dropdown-Liste aus.

2.316 Der Debugger läuft gerade. Beenden?

Ihre Anwendung wird während einer Debugger-Sitzung ausgeführt und wird beendet, wenn Sie auf OK klicken. Klicken Sie, wenn möglich, auf Abbrechen und beenden Sie Ihre Anwendung regulär.

2.317 Typumwandlung

Verwenden Sie dieses Dialogfeld, um für das zu untersuchende Element einen anderen Datentyp anzugeben. Eine Typumwandlung ist nützlich, falls das Inspektorenfenster ein Symbol enthält, für das keine Typinformation vorhanden ist und wenn Sie explizit den Typ für untypisierte Zeiger setzen möchten.

2.318 Eigenschaften Darstellung überwachter Ausdrücke

Start > Ausdruck hinzufügen

Verwenden Sie dieses Dialogfeld zum Hinzufügen eines neuen überwachten Ausdrucks oder zum Ändern der Eigenschaften eines vorhandenen überwachten Ausdrucks. Der überwachte Ausdruck wird in der Liste angezeigt.

Anmerkung: Die im Dialogfeld Darstellung überwachter Ausdrücke aufgeführten Formatbezeichner hängen von denjenigen ab, die vom aktuellen Evaluator unterstützt werden. Nicht alle unten aufgeführten Formatbezeichner stehen für jeden Evaluator zur Verfügung. In den meisten Fällen wird der Evaluator von der Personality angegeben.

Element	Beschreibung
Ausdruck	Gibt den zu überwachenden Ausdruck an. Geben Sie den Ausdruck ein, den Sie überwachen möchten oder bearbeiten Sie ihn. Aus der Dropdown-Liste können Sie einen bereits festgelegten Ausdruck auswählen.
Gruppenname	Gibt die Gruppe an, in die der überwachte Ausdruck aufgenommen werden soll. Wenn Sie eine neue Gruppe festlegen, wird die neue Gruppe hinzugefügt und der überwachte Ausdruck in die neue Gruppe verschoben. Aus der Dropdown-Liste können Sie einen Namen einer bereits vorhandenen Gruppen auswählen.
Wdh-Zähler	Gibt den Wiederholungszähler an, wenn es sich bei dem überwachten Ausdruck um ein Datenelement handelt oder gibt die Anzahl von Elementen in einem Array an, wenn es sich um ein Array handelt. Wenn Sie ein Array überwachen und die Anzahl der Elemente als Wiederholungs-Zähler angeben, zeigt die Liste überwachter Ausdrücke den Wert jedes einzelnen Elements des Arrays an.
Ziffern	Gibt die Anzahl signifikanter Ziffern in einem überwachten Wert an, bei dem es sich um einen Fließkommaausdruck handelt. Geben Sie die Anzahl der Ziffern ein. Diese Option wirkt sich nur aus, wenn Sie als Anzeigeformat Gleitkomma auswählen.
Aktiviert	Aktiviert oder deaktiviert den überwachten Ausdruck. Das Deaktivieren eines überwachten Ausdrucks verbirgt den überwachten Ausdruck vor dem aktuellen Programmlauf. Wenn Sie einen überwachten Ausdruck deaktivieren, bleiben seine Einstellungen definiert, der Debugger wertet den überwachten Ausdruck jedoch nicht aus. Das Deaktivieren überwachter Ausdrücke erhöht die Ausführungsgeschwindigkeit des Debuggers, da er den überwachten Ausdruck nicht prüft, während Sie in Einzelschritten durch das Programm gehen oder es ausführen. Wenn Sie einen überwachten Ausdruck einrichten, ist er per Vorgabe aktiviert.
Funktionsaufrufe zulassen	Wertet den überwachten Ausdruck aus, auch wenn dadurch Funktionen aufgerufen werden. Diese Option ist per Voreinstellung für alle überwachten Ausdrücke deaktiviert. In dieser Einstellung werden Ausdrücke, die Funktionsaufrufe durchführen würden, nicht ausgewertet. Statt dessen wird die Meldung "Nicht verfügbarer Wert" generiert.
Zeichen	Zeigt spezielle Zeichen für die ASCII-Werte 0 bis 31 an (dargestellt als #\\$0, #\\$1F usw.). Dieser Formattyp betrifft Zeichen und Strings.
String	Zeigt die ASCII-Werte 0 bis 31 in der #nn-Notation von Pascal (#\\$0 usw.) an. Dieser Formattyp betrifft Zeichen und Strings.
Dezimal	Zeigt Integer als Dezimalzahlen an. Gilt auch für Integer-Werte in Datenstrukturen. Dieser Formattyp betrifft Integer.

Hexadezimal	Integer werden als Hexadezimalwerte mit dem Präfix 0x (für C++, C#) oder \$ (für Delphi) angezeigt. Gilt auch für Integer-Werte in Datenstrukturen. Dieser Formattyp betrifft Integer.
Fließkomma	Zeigt Integer-Werte in der Gleitkommaschreibweise an (Ganzzahlen oder Zahlen mit Bruchteilen).
Zeiger	Nur für Win32-Anwendungen.
Datensatz/Struktur	Zeigt Feldnamen und -werte wie (X:1;Y:10;Z:5) anstelle von (1,10,5) an.
Standard	Zeigt das Ergebnis in dem Anzeigeformat an, das dem Datentyp des Ausdrucks entspricht. Dieses Format betrifft alle Typen.
Speicherauszug	Nur für Win32-Anwendungen.

Tip: Per Vorgabe zeigt der Debugger das Ergebnis eines überwachten Ausdrucks in dem Format an, das dem Datentyp des Ausdrucks entspricht. Beispielsweise werden Integerwerte in der dezimalen Form angezeigt. Wenn Sie allerdings Hexadezimal für einen Integerausdruck auswählen, wechselt das Anzeigeformat von dezimal zu hexadezimal.

Wenn Sie für ein Element in einer Datenstruktur (wie z.B. ein Array) einen überwachten Ausdruck festlegen, können Sie die Werte der nachfolgenden Datenelemente anzeigen. Angenommen Sie haben ein Array namens xarray mit fünf Integerwerten. Geben Sie die Zahl 5 in das Feld Wdh.-Zähler ein, um alle fünf Werte des Arrays anzuzeigen. Damit ein Wiederholungszähler verwendet werden kann, muss der überwachte Ausdruck jedoch ein einzelnes Datenelement repräsentieren.

Den Wert eines überwachten Ausdrucks ändern Sie im Dialogfeld Auswerten/Ändern.

Tip: Zum Formatieren eines Gleitkomma-Wertes wählen Sie Gleitkomma und geben dann in das Feld Ziffern die Anzahl der signifikanten Nachkommastellen an, die in der Liste überwachter Ausdrücke angezeigt werden sollen.

2.319 Suchen

Suchen▶Suchen

Verwenden Sie dieses Dialogfeld, um den Text anzugeben, den Sie suchen wollen, und um Suchoptionen festzulegen. Der Befehl sucht nach der ersten Quelltextzeile, in der der gesuchte Text vorkommt, und markiert sie.

Element	Beschreibung
Suchen nach	Geben Sie den Suchtext ein oder klicken Sie auf den nach unten gerichteten Pfeil rechts neben dem Eingabefeld, um den Suchtext aus einer Liste früherer Suchtexte auszuwählen.
Groß-/Kleinschreibung	Unterscheidet Groß- und Kleinbuchstaben bei der Suche.
Nur ganze Wörter	Sucht nur nach ganzen Wörtern. (Ist diese Option ausgeschaltet, wird der Suchtext auch in längeren Wörtern aufgefunden.)
Reguläre Ausdrücke	Berücksichtigt reguläre Ausdrücke im Suchtext.
Vorwärts	Sucht von der augenblicklichen Position aus zum Dateiende. Vorwärts ist die Standardeinstellung.
Rückwärts	Sucht von der augenblicklichen Position aus zum Dateianfang.
Gesamter Text	Durchsucht die ganze Datei in der angegebenen Richtung. Gesamter Text ist die Standardeinstellung.
Markierter Text	Durchsucht nur den markierten Text in der angegebenen Richtung. Sie können die Maus oder die Blockbefehle verwenden, um einen Bereich zu markieren.
Ab Cursor	Beginnt bei der aktuellen Cursorposition und sucht dann gemäß der angegebenen Richtung entweder zum Anfang der Datei oder zu deren Ende. Ab Cursor ist die Standardeinstellung für Beginn.
Textanfang	Durchsucht unabhängig von der aktuellen Cursorposition, entweder den ganzen Block oder die ganze Datei, je nach Einstellung unter Bereich.

2.320 In Dateien suchen

Suchen▶In Dateien suchen

Verwenden Sie dieses Dialogfeld, um den Text anzugeben, den Sie suchen wollen, und um Suchoptionen festzulegen. Der Befehl In Dateien suchen arbeitet mit dem Befehl Suche wiederholen des Kontextmenüs des Fensters Meldungen zusammen.

Element	Beschreibung
Groß-/Kleinschreibung	Unterscheidet Groß- und Kleinbuchstaben bei der Suche.
Nur ganze Wörter	Sucht nur nach ganzen Wörtern. Ist diese Option ausgeschaltet, wird der Suchtext auch in längeren Wörtern aufgefunden.
Reguläre Ausdrücke	Berücksichtigt reguläre Ausdrücke im Suchtext.
In allen Dateien des Projekts	Durchsucht alle Dateien des geöffneten Projekts.
In allen geöffneten Dateien	Durchsucht aktuell geöffnete Dateien.
In Verzeichnissen	Stellt die Optionen für Verzeichnissuche zur Verfügung. Die Suche wird in allen angegebenen Dateien durchgeführt.
Dateifilter	Gibt den Pfad für die zu durchsuchenden Dateien an. Verwenden Sie zum Durchsuchen von anderen Dateien Jokerzeichen (wie z.B. *.* oder *.txt) am Ende der Pfadangabe. Zum Festlegen von mehreren Suchmasken trennen Sie die einzelnen Masken durch Semikolons. Um nach Dateien im Stammverzeichnis des Produkts zu suchen, geben Sie das Stammverzeichnis mithilfe der entsprechenden Umgebungsvariable an.
Unterverzeichnisse durchsuchen	Durchsucht auch Unterverzeichnisse der angegebenen Verzeichnisse.
Ergebnisse auf eigener Registerseite anzeigen	Zeigt die Suchergebnisse im Fenster Meldungen auf einer neuen Registerkarte an. Die Registerkarte ist mit Suchen nach <xxx> benannt ist, wobei xxx die gesuchte Zeichenfolge ist. Ist diese Option nicht gewählt, wird eine neue Registerseite nur angelegt, wenn noch keine vorhanden ist. Ist bereits eine vorhanden, werden die Suchergebnisse darin ausgegeben und die Benennung geändert. War die Suche erfolglos, wird die Registerseite gelöscht.

Tip: Jedes Vorkommen des Suchtextes wird in dem Fenster Meldungen am unteren Rand des Quelltext-Editors angezeigt. Doppelklicken Sie auf einen Eintrag in der Liste, um die entsprechende Zeile anzuzeigen.

Tip: Um die letzte Suche zu wiederholen, klicken Sie das Fenster Meldungen mit der rechten Maustaste an und wählen Suche wiederholen.

Tip: Während einer längeren Suche ändert sich die Beschriftung In Dateien suchen; es erscheint der Text Suche in Dateien abbrechen. Um eine laufende Suche zu beenden, öffnen Sie mit der rechten Maustaste die Registerseite für das Suchergebnis und wählen Register schließen oder **Suchen▶Suche in Dateien abbrechen**.

2.321 Überblick zum Suchen von Referenzen

Suchen▶Referenzen suchen

Verwenden Sie dieses Dialogfeld, um Referenzen auf einen markierten Bezeichner zu suchen.

Element	Beschreibung
Entfernen	(Schaltfläche rotes X) Entfernt die markierte Referenz.
Alle entfernen	(Schaltfläche Dokumente mit kleinem x) Entfernt alle Referenzen aus dem Fenster Referenzen suchen. Aufeinander folgende Suchoperationen zeigen so lange eine kumulative Referenzliste an, bis Sie einzelne Referenzen löschen.
+/-	Expandiert bzw. schließt die Knoten in der Referenzhierarchie. Jeder Knoten steht für eine Datei.

2.322 Adresse eingeben

[Suchen](#) [Zu Adresse gehen](#)

Verwenden Sie dieses Dialogfeld, um eine Adresse im Fenster CPU anzuzeigen.

Element	Beschreibung
Eingabefeld	<p>Geben Sie das Symbol, wie z.B. <code>main</code> ein, auf dem das Fenster CPU positioniert werden soll.</p> <p>Alternativ können Sie für verwalteten Code auch eine Adresse im JIT-Compiler-Format (Just-in-Time) eingeben:</p> <p><code>@(Modul-Token, Funktions-Token, Offset)</code></p> <p>Ein Beispiel:</p> <p><code>@(\$3, \$60005C4, \$62)</code></p> <p>Bei unverwaltetem Code können Sie einen linearen 32-Bit-Adresswert eingeben, z.B. <code>\$401018</code>.</p>

2.323 Zu Zeilennummer gehen

Suchen ▶ Zeilennummer

Verwenden Sie dieses Dialogfeld, um im Quelltext-Editor zu einer bestimmten Zeilennummer zu springen.

Element	Beschreibung
Neue Zeilennummer	Geben Sie die Zeilennummer im Quelltext an, zu der Sie springen möchten oder wählen Sie eine Nummer aus der Liste der bereits eingegebenen Zeilennummern aus.

2.324 Suchen und Ersetzen

Suchen▶Ersetzen

Verwenden Sie dieses Dialogfeld, um den Text anzugeben, den Sie suchen und dann durch einen anderen Text (kann auch leer sein) ersetzen wollen.

Element	Beschreibung
Suchen nach	Geben Sie den Suchtext ein oder klicken Sie auf den nach unten gerichteten Pfeil rechts neben dem Eingabefeld, um den Suchtext aus einer Liste früherer Suchtexte auszuwählen.
Ersetzen durch	Geben Sie den Ersatztext ein. Um den Text aus einer Liste früherer Suchtexte auszuwählen, klicken Sie auf den Abwärtspfeil rechts neben dem Eingabefeld. Wenn Sie den Suchtext durch nichts ersetzen wollen, lassen Sie dieses Eingabefeld frei.
Groß-/Kleinschreibung	Unterscheidet Groß- und Kleinbuchstaben bei der Suche.
Nur ganze Wörter	Sucht nur nach ganzen Wörtern. Ist diese Option ausgeschaltet, wird der Suchtext auch in längeren Wörtern aufgefunden.
Reguläre Ausdrücke	Berücksichtigt reguläre Ausdrücke im Suchtext.
Mit Bestätigung	Fragt nach einer Bestätigung vor jedem einzelnen Ersetzen des Suchtextes. Ist diese Option deaktiviert, wird der Suchtext automatisch an allen Stellen im Quelltext-Editor ersetzt.
Vorwärts	Sucht von der augenblicklichen Position aus zum Dateiende. Vorwärts ist die Standardeinstellung.
Rückwärts	Sucht von der augenblicklichen Position aus zum Dateianfang.
Gesamter Text	Durchsucht die ganze Datei in der angegebenen Richtung. Gesamter Text ist die Standardeinstellung.
Markierter Text	Durchsucht nur den markierten Text in der angegebenen Richtung. Sie können die Maus oder die Blockbefehle verwenden, um einen Bereich zu markieren.
Ab Cursor	Beginnt die Suche an der aktuellen Cursorposition und läuft dann gemäß der angegebenen Richtung entweder zum Anfang der Datei oder zu deren Ende.
Textanfang	Durchsucht unabhängig von der aktuellen Cursorposition, entweder den ganzen Block oder die ganze Datei, je nach Einstellung unter Bereich.
Alles ersetzen	Ersetzt alle Vorkommnisse des Suchtextes. Wenn Mit Bestätigung ausgewählt ist, wird bei jedem Auftreten des Suchtextes das Dialogfeld Bestätigen angezeigt.

2.325 Together- Optionen für Sequenzdiagramm-Roundtrips

Tools > Optionen > Together > Verschiedene > Roundtrip-Optionen

Beschreibung der Optionen für Sequenzdiagramm-Roundtrips.

Mit den Optionen der Kategorie **Sequenzdiagramm-Roundtrip** wird das Erzeugen von Sequenzdiagrammen aus Quelltext bzw. das Generieren von Quelltext aus Sequenzdiagrammen gesteuert. Die folgende Tabelle enthält eine Beschreibung dieser Optionen sowie den jeweiligen Vorgabewert.

Option	Beschreibung und Standardwert	
Allgemein		
Vollständige Diagnose anzeigen	<p>Diese Option legt fest, ob beim Erstellen von Sequenzdiagrammen erzeugte Diagnosemeldungen angezeigt werden sollen.</p> <p>Wenn die Option auf True gesetzt ist, kann der Parser detaillierte Fortschrittsmeldungen ausgeben. Der Standardwert ist True.</p>	
Sequenzdiagramm erzeugen		
Tiefe der Aufrufverschachtelung	<p>Dieser Wert legt fest, bis zu welcher Tiefe der Parser die Aufrufsequenz im Quelltext analysiert. Dieser Wert kann verhindern, dass das erzeugte Sequenzdiagramm so groß wird, dass es unbrauchbar wird. Sie können mit Hilfe dieser Option schnell ein Sequenzdiagramm der oberen Ebene für eine komplexe Methode erstellen, indem Sie einen niedrigen Aufrufverschachtelungswert (1-3) angeben. Auf der Grundlage der Ergebnisse Ihres Diagramms können Sie entscheiden, ob Sie eine tiefere Aufrufverschachtelung benötigen und/oder weitere Sequenzdiagramme für alle nicht berücksichtigten Hauptmethoden erstellen.</p> <p>Der Standardwert ist 10.</p>	
Selbstbenachrichtigungen einbeziehen	<p>Diese Option legt fest, ob Selbstbenachrichtigungen in erzeugten Sequenzdiagrammen angezeigt werden sollen. Wenn die Option auf True gesetzt ist, werden Selbstbenachrichtigungen angezeigt. Der Standardwert ist True.</p>	
Quelltext erzeugen		
Vorhandenen Quelltext nie ändern	<p>Wenn diese Option True ist, kann vorhandener Code beim Erzeugen von Quelltext aus einem Sequenzdiagramm nicht geändert werden.</p> <p>Wenn diese Option False ist und bereits Quelltext vorhanden ist, wird beim Erzeugen von Quelltext aus einem Sequenzdiagramm ein Bestätigungsdialogfeld angezeigt, in dem Sie auswählen können, ob der vorhandene Code geändert werden soll.</p> <p>Der Standardwert ist True.</p>	

Siehe auch

Überblick zum Roundtrip Engineering

Konfigurieren von Together (siehe Seite 102)

Optionen (Dialogfeld) (siehe Seite 696)

2.326 Systemmakros

Die folgenden Systemmakros können im Text einiger Optionen verwendet werden:

- **TIME**: aktuelle Uhrzeit
- **LONGTIME**: aktuelle Uhrzeit (langes Format)
- **DATE**: aktuelles Datum
- **LONGDATE**: aktuelles Datum (langes Format)
- **PROJECT**: Projektname
- **DIAGRAM**: Diagrammname
- **USER**: Benutzername
- **COMP**: Computername

Beispiel: Projekt: %PROJECT%, Diagramm: %DIAGRAM%

Folgende Fußzeile wird gedruckt: **Projekt: Projekt1, Diagramm: DgrClass1**

Siehe auch

Konfigurieren von Together (siehe Seite 102)

2.327 Neues Diagramm hinzufügen (Dialogfeld)

Kontextmenü▶Hinzufügen▶Anderes Diagramm

Zum Öffnen dieses Dialogfelds klicken Sie in der Diagrammansicht oder der Modellansicht mit der rechten Maustaste auf den Projektstamm oder ein Namespace-Element und wählen im Kontextmenü **Hinzufügen▶Anderes Diagramm**. Alternativ lässt sich dieses Dialogfeld auch öffnen, indem Sie den Projektstammknoten oder ein Namespace-Element markieren und die Tastenkombination STRG+UMSCHALT+D drücken.

In diesem Dialogfeld werden die Diagramme angezeigt, die dem Projekt hinzugefügt werden können.

Element	Beschreibung
Registerkarte Diagramme	Auf der Registerkarte Diagramme werden die verfügbaren UML-Diagrammtypen angezeigt. Mit Hilfe der Schaltflächen Kleine Symbole und Große Symbole können Sie die Darstellung der UML-Diagrammsymbole ändern. Auf der Registerkarte Diagramme werden standardmäßig große Symbole angezeigt.
Diagrammname	Der ausgewählte Diagrammtyp wird per Voreinstellung im Feld Name angezeigt. Sie können diesen Namen ändern oder einen neuen Namen für das neue Diagramm eingeben. Dieser Name wird in der Modellansicht und auf der Registerkarte Diagramm in der Diagrammansicht angezeigt, wenn das Diagramm für die Bearbeitung geöffnet ist.
Schaltflächen	
Große Symbole	Die Standardeinstellung. Mit dieser Schaltfläche wird die Darstellung der UML-Diagrammsymbole gesteuert. Wenn Sie diese Schaltfläche wählen, werden im Dialogfeld große UML-Diagrammsymbole verwendet.
Kleine Symbole	Mit dieser Schaltfläche wird die Darstellung der UML-Diagrammsymbole gesteuert. Wenn Sie diese Schaltfläche wählen, werden im Dialogfeld kleine UML-Diagrammsymbole verwendet.
OK	Das neue Diagramm des gewählten Typs wird erstellt, in der Diagrammansicht auf einer neuen Registerkarte geöffnet, und das Dialogfeld Neues Diagramm hinzufügen wird geschlossen.
Abbrechen	Alle Änderungen werden verworfen, und das Dialogfeld Neues Diagramm hinzufügen wird geschlossen.
Hilfe	Dieses Hilfethema wird angezeigt.

Siehe auch

Diagramme – Überblick (☞ siehe Seite 1447)

Diagramme erstellen (☞ siehe Seite 116)

2.328 Benutzereigenschaften hinzufügen/entfernen (Dialogfeld)

Kontextmenü▶Benutzereigenschaften

Zum Öffnen dieses Dialogfelds klicken Sie in der Modellansicht oder in der Diagrammansicht mit der rechten Maustaste auf das Diagramm oder Modellelement und wählen **Benutzereigenschaften**.

Dieses Dialogfeld dient zur Erstellung von Benutzereigenschaften und OCL-Einschränkungen. Es enthält eine Liste der vorhandenen Eigenschaften. Jeder Eintrag in der Liste besteht aus dem Paar *Name-Wert*.

Sie können Einträge hinzufügen oder löschen und Namen sowie Werte bearbeiten. Der Name kann im Textfeld *Name* geändert werden. Zur Bearbeitung von Werten können Sie entweder das Textfeld verwenden oder auf die entsprechende Schaltfläche klicken und den Text im Editorfenster eingeben.

Hinzufügen	In der Liste der Eigenschaften wird ein neuer Eintrag erzeugt.
Entfernen	Der ausgewählte Eintrag wird aus der Liste der Eigenschaften entfernt.
OK	Die Änderungen werden gespeichert, und das Dialogfeld wird geschlossen.
Abbrechen	Die Änderungen werden verworfen, und das Dialogfeld wird geschlossen.
Hilfe	Dieses Thema wird angezeigt.

Siehe auch

Überblick zur OCL-Unterstützung (☞ siehe Seite 1453)

Benutzereigenschaften verwenden (☞ siehe Seite 135)

2.329 Parameter ändern (Dialogfeld)

Refactor ► Parameter ändern

Das Dialogfeld Parameter ändern kann über das Hauptmenü Refactoring oder durch Auswahl des Befehls Refactoring ► Parameter ändern im Kontextmenü von Methoden geöffnet werden.

Anmerkung: Diese Funktion steht nur für Implementierungsprojekte zur Verfügung.

Klasse	Dieses schreibgeschützte Feld enthält den Namen der Klasse, in der sich die Methode befindet.
Methode	In diesem schreibgeschützten Feld werden das ausgewählte Member und seine aktuellen Parameter (sofern vorhanden) angezeigt.
Member auswählen	In einer Tabelle werden alle vorhandenen Parameter sowie alle neuen Parameter angezeigt, die Sie der Methode hinzufügen. Die Abfolge der Parameter in der Tabelle entspricht der Reihenfolge der Parameter in der Methode. Mit den Schaltflächen Hinzufügen und Entfernen können Sie Parameter für eine Methode hinzufügen oder entfernen. Wenn Sie einen neuen Parameter hinzufügen, können Sie seinen Typ, Namen und Vorgabewert ändern. Bei einem vorhandenen Parameter kann der Name geändert werden. Mit den Schaltflächen Nach oben und Nach unten können Sie die Reihenfolge der Parameter ändern.
Verwendungsvorschau	Die Option Verwendungsvorschau ist standardmäßig aktiviert. Wenn Sie diese Option aktivieren und dann auf OK klicken, wird das Fenster Refactoring geöffnet, und Sie können die Refactoring-Operation vor der Übernahme überprüfen. Wenn Sie bei deaktivierter Option auf OK klicken, wird das Fenster Refactoring mit den abgeschlossenen Änderungen geöffnet.
Schaltflächen	
Hinzufügen	Der Methode wird ein neuer Parameter hinzugefügt.
Entfernen	Der ausgewählte Parameter wird aus der Methode entfernt.
Nach oben	Der ausgewählte Parameter wird um eine Zeile nach oben verschoben.
Nach unten	Der ausgewählte Parameter wird um eine Zeile nach unten verschoben.
OK	Das Fenster Refactoring wird geöffnet.
Abbrechen	Die Änderungen werden verworfen, und das Dialogfeld wird geschlossen.

Siehe auch

Überblick zum Refactoring (siehe Seite 1456)

Ändern von Methodenparametern (siehe Seite 103)

2.330 Zielprojekt auswählen/Quellprojekt auswählen (Dialogfeld)

Kontextmenü der **Modellansicht**►**In Quelltext umwandeln (oder: Quelltext aus Designprojekt umwandeln)**

Dieses Dialogfeld wird über das Kontextmenü eines Projekts in der Modellansicht aufgerufen. Wählen Sie das gewünschte Zielprojekt aus, und klicken Sie auf Umwandeln.

Element	Beschreibung
Vorhandene Projekte	Zeigt die Projekte in der aktuellen Projektgruppe an. In Implementierungsprojekten sind die Designprojekte grau dargestellt (nicht verfügbar). In Designprojekten sind die Implementierungsprojekte grau dargestellt (nicht verfügbar).
Namenszuordnungsdateien für Quelltexterzeugung	Über dieses Kontrollkästchen wird die Funktion für die Namenszuordnung aktiviert oder deaktiviert.
Umwandeln	Klicken Sie auf diese Schaltfläche, um die Umwandlung des Designprojekts in Quelltext zu starten. Die Schaltfläche steht erst zur Verfügung, nachdem in der Liste ein gültiges Quelltextprojekt ausgewählt wurde.
Abbrechen	Ihre Auswahlen werden verworfen, und das Dialogfeld wird geschlossen.
Hilfe	Dieses Thema wird geöffnet.

Siehe auch

Überblick zur Umwandlung in Quelltext (☞ siehe Seite 1452)

Umwandeln eines Designprojekts in Quelltext (☞ siehe Seite 264)

2.331 Bearbeiten der Hyperlinks für Diagramm (Dialogfeld)

Kontextmenü▶**Hyperlinks**▶**Bearbeiten**

Dieses Dialogfeld wird über die Kontextmenüs im Diagrammeditor oder in der Modellansicht geöffnet. Es enthält zwei Registerkarten, auf denen Sie Hyperlinks zu Modellelementen und externen Ressourcen erstellen können.

Dialogtitel	Der Titel des Dialogfelds variiert und entspricht dem aufrufenden Objekt.
Registerkarte Modellelemente	Im Fenster auf der linken Seite des Dialogfelds wird der Inhalt angezeigt, der für das Projekt verfügbar ist. Navigieren Sie im Explorer zu einem Element, und wählen Sie es aus, um es in die Werteliste für das aufrufende Objekt zu übernehmen.
Registerkarte Externe Dokumente	Im Fenster Zuletzt verwendete Dokumente werden die externen Inhalte angezeigt, die Sie für das Projekt bereitstellen. Die betreffenden Elemente können durch Dateisystemressourcen oder URLs repräsentiert werden. Mit Hilfe der Schaltflächen Durchsuchen und URL können Sie diese Ressourcen angeben.
Durchsuchen	Klicken Sie auf diese Schaltfläche, um ein Dialogfeld zur Dateiauswahl zu öffnen. Navigieren Sie zur gewünschten Datei, und klicken Sie auf OK .
URL	Wenn Sie auf diese Schaltfläche klicken, wird das Dialogfeld Dokument-URL geöffnet. Geben Sie einen URL in das Textfeld ein, und klicken Sie auf OK .
Leeren	Durch Klicken auf diese Schaltfläche werden alle Einträge aus der Liste im Fenster Zuletzt verwendete Dokumente entfernt.
Fenster Ausgewählte	In diesem Fenster werden zwei Arten von Daten angezeigt. Werte, die bereits existieren und vom aufrufenden Objekt übergeben wurden (sofern vorhanden). Werte, die aus dem Fenster auf der linken Seite übernommen wurden (sofern vorhanden).

Schaltflächen	
Hinzufügen	Ist aktiviert, wenn ein Element im linken Fenster ausgewählt wird. Wenn Sie auf die Schaltfläche klicken, wird das markierte Element der Liste im rechten Fenster hinzugefügt.
Entfernen	Ist aktiviert, wenn ein Eintrag im rechten Fenster ausgewählt wird. Klicken Sie auf die Schaltfläche, um das Element zu entfernen. Alle entfernten Werte oder Objekte werden endgültig aus der aufrufenden Eigenschaft bzw. dem Diagramm gelöscht, wenn Sie auf OK klicken.
Alle entfernen	Ist aktiviert, wenn Einträge im rechten Fenster vorhanden sind. Klicken Sie auf die Schaltfläche, um alle Elemente aus diesem Fenster zu entfernen. Alle entfernten Werte oder Objekte werden endgültig aus der aufrufenden Eigenschaft bzw. dem Diagramm gelöscht, wenn Sie auf OK klicken.

Siehe auch

Überblick zu Hyperlinks (siehe Seite 1450)

Verknüpfen von Diagrammen durch Hyperlinks (siehe Seite 125)

2.332 Diagramm als Bild exportieren (Dialogfeld)

Datei ▶ Diagramm als Bild exportieren

Mit Hilfe dieses Dialogfelds können Sie das aktive Diagramm im festgelegten Format speichern. Zum Öffnen des Dialogfelds stellen Sie in der Diagrammansicht den Fokus auf das zu exportierende Diagramm und wählen **Datei | Diagramm als Bild exportieren**.

Zoom	In diesem Bereich legen Sie den Zoom-Faktor und die Größe des Bildes fest.
Z	Geben Sie einen Zoom-Faktor an.
W	Geben Sie die Breite des Bildes in Pixel an.
H	Geben Sie die Höhe des Bildes in Pixel an.
Vorschau	Klicken Sie auf den nach unten weisenden Pfeil, um die Seite Vorschau anzuzeigen.
Vorschau-Zoom	Verwenden Sie diesen Schieberegler, um den Zoom-Faktor für die Vorschau festzulegen. Der aktuelle Zoom-Faktor wird links neben dem Schieberegler angezeigt.
Automatischer Vorschau-Zoom	Aktivieren Sie diese Option, wenn das Bild in das Vorschaufenster eingepasst werden soll.
Speichern	Klicken Sie auf diese Schaltfläche, um das Dialogfeld zum Speichern zu öffnen. Geben Sie den Namen der Zieldatei, den Zielspeicherort und das Format des exportierten Bildes an.

Siehe auch

Funktionen für Import und Export (☞ siehe Seite 1459)

Exportieren eines Diagramms in eine Bilddatei (☞ siehe Seite 136)

2.333 Interface extrahieren/Superklasse extrahieren (Dialogfelder)

Refactor ▶ Superklasse extrahieren (oder: Interface extrahieren)

Die Dialogfelder **Interface extrahieren** und **Superklasse extrahieren** können über das Hauptmenü **Refactoring** oder durch Auswahl des Befehls **Refactoring | Superklasse extrahieren** (bzw. **Interface extrahieren**) im Kontextmenü bestimmter Klassendiagrammelemente geöffnet werden. Der Befehl **Interface extrahieren** steht für Klassen, Strukturen, Methoden, Eigenschaften, Ereignisse und Indizierer zur Verfügung. Der Befehl **Superklasse extrahieren** ist für Klassen, Interfaces (**Superinterface extrahieren**), Methoden, Eigenschaften, Ereignisse, Felder und Indizierer verfügbar.

Anmerkung: Diese Funktion steht nur für Implementierungsprojekte zur Verfügung.

Name Interface/Name Superklasse	des der	Geben Sie den Namen des zu erstellenden Interface oder der zu erstellenden Superklasse ein.
Namespace		In diesem Feld wird der Namespace festgelegt, in dem sich das Interface bzw. die Superklasse befinden soll. Geben Sie den voll qualifizierten Namen für den Namespace ein, oder klicken Sie auf die entsprechende Schaltfläche, und wählen Sie den gewünschten Ziel-Namespace aus.
Member auswählen		In einer Tabelle werden die Member angezeigt, die Sie für das Extrahieren in das neue Interface oder die neue Superklasse auswählen können. Standardmäßig sind alle gefundenen Member ausgewählt. Aktivieren Sie in der ersten Spalte der Tabelle die Kontrollkästchen der Member, die extrahiert werden sollen.
Vor Refactoring Referenzen anzeigen		Die Option Vor Refactoring Referenzen anzeigen ist standardmäßig aktiviert. Wenn Sie bei aktiverter Option auf OK klicken, wird das Fenster Refactoring geöffnet, in dem Sie die Refactoring-Operation vor der Übernahme überprüfen können. Wenn Sie bei deaktiverter Option auf OK klicken, wird das Fenster Refactoring mit den extrahierten Informationen geöffnet.
Schaltflächen		
OK		Das Fenster Refactoring wird geöffnet.
Abbrechen		Alle Änderungen werden verworfen, und das Dialogfeld wird geschlossen.

Siehe auch

Überblick zum Refactoring (☞ siehe Seite 1456)

Extrahieren von Interfaces und Superklassen (☞ siehe Seite 104)

2.334 Methode extrahieren (Dialogfeld)

Refactor ▶ Methode extrahieren

Das Dialogfeld Methode extrahieren kann über das Hauptmenü Refactoring oder durch Auswahl des Befehls Refactoring ▶ Methode extrahieren für mehrere abgeschlossene Anweisungen im RAD Studio-Editor geöffnet werden.

Anmerkung: Diese Funktion steht nur für Implementierungsprojekte zur Verfügung.

Hinweis	Dieser Befehl ist nur im RAD Studio-Editor verfügbar.
Name	Geben Sie den Namen der Methode ein, die aus dem ausgewählten Quelltextfragment erstellt werden soll.
Sichtbarkeit	Wählen Sie einen Sichtbarkeitsmodifizierer in der Dropdown-Liste aus. Folgende Werte sind verfügbar: public, protected, private, internal, internal protected
Einleitender Kommentar	Geben Sie einen Quelltextkommentar mit einer Beschreibung der neuen Methode ein.
Statisch	Aktivieren Sie gegebenenfalls das Kontrollkästchen, um das Feld Statisch auszuwählen.

Siehe auch

Überblick zum Refactoring (siehe Seite 1456)

Methoden extrahieren (siehe Seite 105)

2.335 Dokumentation erzeugen (Dialogfeld)

Tools ▶ Dokumentation erzeugen

Together bietet einen UML-Dokumentationsexperten, mit dem Sie HTML-Dokumentation für Ihre Projekte erzeugen können. Um dieses Dialogfeld zu öffnen, klicken Sie im Hauptmenü auf **[Tools | Dokumentation erzeugen]**.

Bereich	Sie können den Bereich, für den die Dokumentation erstellt werden soll, eingrenzen, indem Sie die entsprechende Bereichsoption auswählen. Mit den Optionsfeldern unter <i>Bereich</i> (im oberen Teil des Dialogfelds) geben Sie an, welche Teile des Projekts analysiert und in die Dokumentationserzeugung einbezogen werden.
Aktueller Namespace	Die Dokumentationserzeugung erfolgt nur für den aktuellen Namespace, der in der Modellansicht oder Diagrammansicht ausgewählt ist.
Aktueller Namespace mit abgeleiteten Namespaces	Die Dokumentationserzeugung erfolgt für den aktuellen Namespace, der in der Modellansicht ausgewählt ist, sowie für alle von ihm abgeleiteten Namespaces.
Aktuelles Diagramm	Die Dokumentationserzeugung erfolgt für das aktuelle Diagramm, das in der Diagrammansicht den Fokus besitzt.
Alles	Die Dokumentationserzeugung erfolgt für das gesamte Projekt.
Optionen	Der Bereich <i>Optionen</i> des Dialogfelds enthält Einstellungen, mit denen Sie das Ziel sowie weitere Aktionen festlegen können:
Ausgabeordner	Geben Sie den Speicherort für die erzeugte Ausgabe ein, oder wählen Sie ihn mit Hilfe der Dateiauswahl.
Diagramme einbeziehen	Wenn Sie dieses Kontrollkästchen aktivieren, enthält die Ausgabe Diagrammbilder.
Navigationshierarchie einbeziehen	Ist dieses Kontrollkästchen aktiviert, enthält die Ausgabe eine Navigationshierarchie.
HTML-Browser aufrufen	Aktivieren Sie dieses Kontrollkästchen, um die Dokumentation in den externen Web-Browser zu laden.
Hinweis	Der Navigations-Frame in der Dokumentation funktioniert nur, wenn JDK/JRE 1.4 installiert und im Browser und aktiviert ist.
Schaltflächen	
OK	Die Eingaben werden übernommen, und die Erzeugung der Dokumentation wird gestartet.
Abbrechen	Die Eingaben werden verworfen, das Dialogfeld wird geschlossen, und die Dokumentation wird nicht generiert.

Siehe auch

Projektdokumentation erzeugen – Überblick (siehe Seite 1458)

Projektdokumentation erzeugen (siehe Seite 224)

2.336 Inline für Variable (Dialogfeld)

Refactor ► Inline für Variable

In diesem Dialogfeld wird die Anzahl der Variablenvorkommen angezeigt, auf die der Befehl **Inline für Variable** angewendet wird. Klicken Sie auf **OK**, um die Änderungen abzuschließen.

Das Dialogfeld **Inline für Variable** kann über das Hauptmenü **Refactoring** oder durch Auswahl des Befehls **Refactoring | Inline für Variable** für eine lokale Variable im Delphi-Quelltext-Editor geöffnet werden.

Anmerkung: Die ausgewählte Variable darf später im Quelltext nicht geändert werden. Wenn dies doch der Fall ist, wird die Fehlermeldung *Auf Variable wird zum Schreiben nicht zugegriffen* angezeigt.

Anmerkung: Diese Funktion steht nur für Implementierungsprojekte zur Verfügung.

Anmerkung: Dieser Befehl ist nur im Delphi-Quelltext-Editor, nicht aber in der Diagrammansicht verfügbar.

Schaltflächen	
OK	Die Inline-Variable wird erzeugt, und das Dialogfeld wird geschlossen.
Abbrechen	Die Änderungen werden verworfen, und das Dialogfeld wird geschlossen.

Siehe auch

Überblick zum Refactoring (☞ siehe Seite 1456)

Inline-Variablen erstellen (☞ siehe Seite 107)

2.337 Feld einführen (Dialogfeld)

Refactor ▶ Feld einführen

Das Dialogfeld **Feld einführen** kann über das Hauptmenü **Refactoring** oder durch Auswahl des Befehls **Refactoring | Feld einführen** für einen Ausdruck im Delphi-Quelltext-Editor geöffnet werden.

Anmerkung: Diese Funktion steht nur für Implementierungsprojekte zur Verfügung.

Anmerkung: Dieser Befehl ist nur im Delphi-Quelltext-Editor verfügbar.

Name	Geben Sie einen Namen für das neue Feld ein.
Sichtbarkeit	Legen Sie die Sichtbarkeit des neuen Feldes fest. Im Kombinationsfeld stehen folgende Auswahlmöglichkeiten zur Verfügung: public , protected , private , internal und internal protected .
Initialisieren	Geben Sie an, wo das neue Feld initialisiert werden soll. Im Kombinationsfeld stehen folgende Auswahlmöglichkeiten zur Verfügung: Aktuelle Methode , Klassenkonstruktor(en) und Felddeklaration .
Statisch	Aktivieren Sie dieses Kontrollkästchen bei Bedarf.
Alle Vorkommen ersetzen	Aktivieren Sie dieses Kontrollkästchen, wenn alle Vorkommen des Ausdrucks ersetzt werden sollen.
Schaltflächen	
OK	Das neue Feld wird erzeugt, und das Dialogfeld wird geschlossen.
Abbrechen	Die Änderungen werden verworfen, und das Dialogfeld wird geschlossen.

Siehe auch

Überblick zum Refactoring (☞ siehe Seite 1456)

Felder einführen (☞ siehe Seite 108)

2.338 Variable einführen (Dialogfeld)

Refactor ▶ Variable einführen

Das Dialogfeld **Variable einführen** kann über das Hauptmenü **Refactoring** oder durch Auswahl des Befehls **Refactoring | Variable einführen** für eine Variable im Delphi-Quelltext-Editor geöffnet werden.

Anmerkung: Diese Funktion steht nur für Implementierungsprojekte zur Verfügung.

Anmerkung: Dieser Befehl ist nur im Delphi-Quelltext-Editor verfügbar.

Name	Geben Sie den Namen für die neue Variable ein. Die neue Variable ist vom selben Typ wie die ursprüngliche Variable.
Alle Vorkommen ersetzen	Aktivieren Sie dieses Kontrollkästchen, wenn alle Vorkommen des Ausdrucks ersetzt werden sollen. Im Dialogfeld wird die Anzahl der Variablenvorkommen angezeigt, die durch die Operation ersetzt werden. Beachten Sie aber, dass keine Vorkommen vor der Stelle im Quelltext ersetzt werden, an der Sie die neue Variable einführen.
Schaltflächen	
OK	Die neue Variable wird erzeugt, und das Dialogfeld wird geschlossen.
Abbrechen	Die Änderungen werden verworfen, und das Dialogfeld wird geschlossen.

Siehe auch

Überblick zum Refactoring (☞ siehe Seite 1456)

Einführen neuer Variablen (☞ siehe Seite 109)

2.339 Modellunterstützung

Projekt ▶ Together-Unterstützung

Verwenden Sie dieses Dialogfeld, um die Together-Modellunterstützung für die aktuell geöffneten Projekte zu aktivieren bzw. zu deaktivieren.

Element	Beschreibung
Projektliste	Zeigt die Projekte in der aktuellen Projektgruppe an.

Siehe auch

Together-Unterstützung für Projekte aktivieren (↗ siehe Seite 248)

2.340 Verlagern (Dialogfeld)

Refactor ▶ Verlagern

Das Dialogfeld **Verlagern** kann mit dem Befehl **Verlagern** im Menü **Refactoring** oder mit dem Befehl **Refactoring | Verlagern** im Kontextmenü statischer Member (d.h. statischer Methoden, Felder oder Eigenschaften) geöffnet werden.

Anmerkung: Diese Funktion steht nur für Implementierungsprojekte zur Verfügung.

Member verlagern	Dieses Feld enthält die Liste der ausgewählten statischen Member. Sie können mehrere statische Member gleichzeitig verlagern. Um statische Member auszuwählen oder ihre Auswahl aufzuheben, aktivieren bzw. deaktivieren Sie das Kontrollkästchen neben dem Member-Namen.
Nach (voll qualifizierter Name des Namespace)	In diesem Feld wird die Klasse festgelegt, in der sich das bzw. die statischen Member befinden sollen. Geben Sie den voll qualifizierten Namen für die Klasse ein, oder klicken Sie auf die Schaltfläche Durchsuchen , und wählen Sie die Klasse aus.
Verwendungsvorschau	Die Option Verwendungsvorschau ist standardmäßig aktiviert. Wenn Sie bei aktiverter Option auf OK klicken, wird das Fenster Refactoring geöffnet, in dem Sie die Refactoring-Operation vor der Übernahme überprüfen können. Ist die Option deaktiviert, wird die Verlagerung unmittelbar nach dem Klicken auf OK durchgeführt, und die Ergebnisse werden im Fenster Refactoring angezeigt.
Schaltflächen	
OK	Das Fenster Refactoring wird geöffnet.
Abbrechen	Die Änderungen werden verworfen, und das Dialogfeld wird geschlossen.

Siehe auch

Überblick zum Refactoring (↗ siehe Seite 1456)

Verlagern von Quelltextelementen (↗ siehe Seite 110)

2.341 Together-Optionen (Dialogfeld)

Tools > Optionen

Im Dialogfeld Optionen sind die Konfigurationsoptionen nach Kategorien in einer Baumstruktur angeordnet. Wenn Sie eine Kategorie auswählen, werden die zugehörigen Konfigurationsoptionen angezeigt. Um dieses Dialogfeld zu öffnen, klicken Sie im Hauptmenü auf **Tools > Optionen**. Wählen Sie in der Baumstruktur auf der linken Seite des Dialogfelds den Ordner Together aus.

Im unteren Bereich des Dialogfelds wird eine Beschreibung der ausgewählten Option angezeigt.

Die folgenden Optionskategorien befinden sich in der Baumstruktur unterhalb der Konfigurationsebenen:

Allgemein	Mit den Optionen der Kategorie Allgemein können Sie bestimmte Verhaltensweisen der Benutzeroberfläche anpassen, die nicht durch eine andere Optionskategorie (etwa die Kategorie für Diagramme) abgedeckt werden.
Diagramm	Mit den Optionen der Kategorie Diagramm werden verschiedene Aspekte des Standardverhaltens und der Standarddarstellung von Diagrammen gesteuert.
Erscheinungsbild	
Layout	
Drucken	
Ansichtsverwaltung	
Dokumentation generieren	Mit den Optionen der Kategorie Dokumentation generieren wird der Inhalt (und dessen Darstellung) festgelegt, der in die erzeugte HTML-Dokumentation aufgenommen werden soll.
Modellansicht	Mit den Optionen der Kategorie Modellansicht steuern Sie, wie der Diagramminhalt im Fenster Modellansicht dargestellt wird.
Quelltext	Mit diesen Optionen können verschiedene LiveSource-Parameter gesteuert werden.

Schaltflächen	
OK	Die Änderungen werden übernommen, und das Dialogfeld wird geschlossen.
Abbrechen	Die Änderungen werden verworfen, und das Dialogfeld wird geschlossen.
Hilfe	Die Online-Hilfe von RAD Studio wird angezeigt.

Siehe auch

- Konfigurieren von Together ([siehe Seite 102](#))
- Konfigurationsebenen ([siehe Seite 1280](#))
- Optionswert-Editoren ([siehe Seite 1280](#))
- Allgemeine Optionen ([siehe Seite 1281](#))
- Diagrammoptionen (Erscheinungsbild) ([siehe Seite 1282](#))
- Diagrammoptionen (Layout) ([siehe Seite 1284](#))
- Diagrammoptionen (Drucken) ([siehe Seite 1287](#))

Diagrammoptionen (Ansichtsfilter) (siehe Seite 1289)

Optionen für die Dokumentationserzeugung (siehe Seite 1291)

Optionen für die Modellansicht (siehe Seite 1293)

Optionen für Sequenzdiagramm-Roundtrips (siehe Seite 680)

Quelltextoptionen (siehe Seite 1294)

2.342 Audit drucken (Dialogfeld)

Fenster Audit-Ergebnisse Schaltfläche Drucken

In diesem Dialogfeld können Sie Gruppen mit Audit-Ergebnissen (Audit-Sets) zum Drucken auswählen und an einen Drucker senden. Das Dialogfeld wird aus der Ansicht mit dem Audit-Ergebnisbericht aufgerufen.

Ansicht auswählen	<p>In diesem Listenfeld wählen Sie den Bereich der zu druckenden Ergebnisse aus. Audit-Ergebnisse werden in Form von Registerseiten in der Ansicht mit dem Audit-Ergebnisbericht angezeigt. Das Kontextmenü des Ergebnisberichts enthält den Befehl Gruppieren nach, mit dem Sie die Ergebnisse gruppieren oder ihre Gruppierung aufheben können.</p> <p>Die Option Aktive Gruppe steht nur zur Verfügung, wenn die Ergebnisse mit dem Befehl Gruppieren nach gruppiert wurden. Folgende Ansichtsoptionen stehen zur Wahl:</p> <p>Alle Ergebnisse: Wenn die Ergebnisse gruppiert sind, wird bei Auswahl dieser Option ein Bericht für alle Gruppen der aktuellen Registerseite ausgegeben. Sind die Ergebnisse nicht gruppiert, werden alle Ergebnisse für die aktuelle Registerseite gedruckt.</p> <p>Aktive Gruppe: Wenn die Ergebnisse gruppiert sind, können Sie eine Gruppe auf der aktuellen Registerseite auswählen und einen Bericht für diese Gruppe drucken lassen.</p> <p>Ausgewählte Zeilen: Sie können in der Ansicht mit dem Audit-Ergebnisbericht eine oder mehrere Zeilen auswählen. Bei Auswahl von Ausgewählte Zeilen wird ein Bericht für diese Zeilen generiert.</p>
Druck-Zoom	Geben Sie den Zoom-Faktor für den Druck ein. Die Standardeinstellung für den Zoom-Faktor ist 1.
An Seitengröße anpassen	Aktivieren Sie diese Option, wenn die Ergebnisse auf eine einzelne Seite gedruckt werden sollen. Das Feld Druck-Zoom steht dann nicht zur Verfügung.
Vorschau	Klicken Sie auf den nach unten weisenden Pfeil, um die Seite Vorschau anzuzeigen.
Vorschau-Zoom	Stellen Sie den Vorschau-Zoom mit Hilfe des Schiebereglers Vorschau-Zoom (autom.) ein. Der aktuelle Zoom-Faktor wird links neben dem Schieberegler angezeigt.
Autom. Zoom der Vorschau	Aktivieren Sie diese Option, wenn das Bild in das Vorschaufenster eingepasst werden soll.
Schaltflächen	
Drucken	Klicken Sie auf diese Schaltfläche, um den gewählten Audit-Bericht an den Standarddrucker zu senden. Mit dem nach unten weisenden Pfeil können Sie den Befehl Druckdialog auswählen und das Dialogfeld Drucken öffnen, in dem sich die Druckoptionen konfigurieren lassen.
Abbrechen	Das Dialogfeld wird geschlossen, und der Audit-Bericht wird nicht gedruckt.
Hilfe	Diese Seite wird geöffnet.

Siehe auch

[Audit-Ergebnisse anzeigen](#) (siehe Seite 270)

[Audit drucken \(Dialogfeld\)](#)

[Audit-Ergebnisse \(Fenster\)](#) (siehe Seite 1273)

2.343 Diagramm drucken (Dialogfeld)

Datei ▶ Drucken

In diesem Dialogfeld können Sie Diagramme zum Drucken auswählen und an einen Drucker senden. Zum Öffnen des Dialogfelds wählen Sie im Hauptmenü **Datei | Drucken**, während ein Diagramm in der Diagrammansicht geöffnet ist.

Diagramme drucken	Wählen Sie die zu druckenden Diagramme in diesem Listenfeld aus. Folgende Optionen stehen zur Wahl: Aktives Diagramm Aktives mit benachbarten (alle Diagramme innerhalb desselben Namespace) Alle geöffneten Diagramme Alle Diagramme im Modell
Druck-Zoom	Geben Sie einen Zoom-Faktor für den Ausdruck an. Die Standardeinstellung für den Zoom-Faktor ist 1.
An Seitengröße anpassen	Markieren Sie diese Option, wenn das Diagramm auf eine einzige Seite gedruckt werden soll. Das Feld Druck-Zoom steht dann nicht zur Verfügung.
Vorschau	Klicken Sie auf den nach unten weisenden Pfeil, um die Seite Vorschau anzuzeigen.
Vorschau-Zoom	Verwenden Sie diesen Schieberegler, um den Zoom-Faktor für die Vorschau festzulegen. Der aktuelle Zoom-Faktor wird links neben dem Schieberegler angezeigt.
Autom. Zoom der Vorschau	Aktivieren Sie diese Option, wenn das Bild in das Vorschaufenster eingepasst werden soll.
Drucken	Klicken Sie auf diese Schaltfläche, um die ausgewählten Diagramme auf dem Standarddrucker auszugeben. Mit dem nach unten weisenden Pfeil können Sie den Befehl Druckdialog auswählen und das Dialogfeld Drucken öffnen, in dem sich die Druckoptionen konfigurieren lassen.

Siehe auch

Diagramme drucken (siehe Seite 143)

2.344 Member in übergeordnete Klasse verschieben/Member in abgeleitete Klasse verschieben (Dialogfelder)

Refactor ▶ Member in übergeordnete Klasse verschieben (oder: Member in abgeleitete Klasse verschieben)

Sie können die Dialogfelder **Member in übergeordnete Klasse verschieben** und **Member in abgeleitete Klasse verschieben** über das Hauptmenü **Refactoring** oder durch Auswahl des Befehls **Refactoring | Member in übergeordnete Klasse verschieben** (bzw. **Member in abgeleitete Klasse verschieben**) im Kontextmenü bestimmter Klassendiagrammelemente öffnen. Die Befehle **Member in übergeordnete Klasse verschieben/Member in abgeleitete Klasse verschieben** stehen für Methoden, Eigenschaften, Felder, Indizierer und Ereignisse zur Verfügung.

Warnung: Diese Funktion steht nur für Implementierungsprojekte zur Verfügung.

Member auswählen:	In einer Tabelle werden die Member aufgeführt, die Sie für das Verschieben in die übergeordnete bzw. abgeleitete Klasse ausgewählt haben. Standardmäßig sind alle Member ausgewählt. Aktivieren Sie in der ersten Spalte der Tabelle die Kontrollkästchen der Member, die verschoben werden sollen. In der dritten Spalte können Sie angeben, ob das Member <i>abstract</i> sein soll.
Klasse auswählen, in die das Member verschoben werden soll:	Im unteren Bereich des Dialogfelds wird eine Hierarchiestruktur angezeigt. Wählen Sie hier die Klasse aus, in die die ausgewählten Member verschoben werden sollen.
Vor Refactoring Referenzen anzeigen:	Die Option Vor Refactoring Referenzen anzeigen ist standardmäßig aktiviert. Wenn Sie bei aktiverter Option auf OK klicken, wird das Fenster Refactoring geöffnet, in dem Sie die Refactoring-Operation vor der Übernahme überprüfen können. Ist die Option deaktiviert, wird das Verschieben in die übergeordnete bzw. abgeleitete Klasse unmittelbar nach dem Klicken auf OK durchgeführt, und die Ergebnisse werden im Fenster Refactoring angezeigt.
Schaltflächen	
OK	Das Fenster Refactoring wird geöffnet.
Abbrechen	Die Änderungen werden verworfen, und das Dialogfeld wird geschlossen.

Siehe auch

Überblick zum Refactoring (siehe Seite 1456)

Member in übergeordnete oder abgeleitete Klasse verschieben (siehe Seite 111)

2.345 QS-Audits (Dialogfeld)

Kontextmenü▶QS-Audits

Wählen Sie zum Öffnen des Dialogfelds **Audits** den Befehl **QS-Audits** im Kontextmenü der Diagrammansicht, Modellansicht oder der Klasse bzw. des Interface.

Warnung: Diese Funktion steht nur für Implementierungsprojekte zur Verfügung.

Symbolleiste	Mit den Schaltflächen in der Symbolleiste des Dialogfelds Audits werden benutzerdefinierte Audit-Sets geladen und gespeichert sowie die Audits festgelegt, die für die Projekte ausgeführt werden sollen. In der folgenden Tabelle sind die Schaltflächen der Symbolleiste beschrieben.
Schaltfläche	Beschreibung
Set laden	Ein Fenster zur Dateiauswahl wird geöffnet, in dem eine .adt-Datei mit einem benutzerdefinierten Audit-Set geladen werden kann.
Set speichern unter	Ein Dialogfeld zum Speichern wird geöffnet. Geben Sie den Namen und den Speicherort für die .adt-Datei ein, in der das aktuelle Audit-Set gespeichert werden soll.
Alles auswählen	Die Kontrollkästchen aller Audits im Dialogfeld werden aktiviert.
Auswahl aufheben	Die Auswahl der Kontrollkästchen aller Audits im Dialogfeld wird aufgehoben.
Standard setzen	Das Standard-Set (in der Datei default.adt) wird wiederhergestellt.
Audit suchen	Es wird zu dem Audit gewechselt, dessen Name mit dem angegebenen String beginnt.
Bereich	In diesem Feld legen Sie fest, für welche Teile des Projekts die Audits ausgeführt werden. Klicken Sie auf den Abwärtspfeil, und wählen Sie die aktuelle Auswahl oder das gesamte Modell als Bereich. Bei der Option Auswahl werden Audits nur für das Diagramm, den Namespace oder die Klasse ausgeführt, die Sie vor dem Öffnen des Dialogfelds Audits ausgewählt haben. Wenn Sie Modell wählen, werden die Audits für das gesamte Projekt ausgeführt.
Anmerkung:	Wenn in der Diagrammansicht oder der Modellansicht keine Elemente ausgewählt sind, wird die Option Bereich automatisch auf Modell gesetzt. Um bestimmte Klassen, Namespaces oder Diagramme zu überprüfen, müssen Sie diese daher auswählen, bevor Sie das Dialogfeld Audits öffnen.

Auswahlfenster Das Auswahlfenster enthält eine Liste der verfügbaren Audits. Die Audits sind nach Kategorien sortiert. Aktivieren Sie die Kontrollkästchen der Audits, die ausgeführt werden sollen. Wenn Sie alle Audits einer Kategorie auswählen oder ihre Auswahl aufheben möchten, aktivieren bzw. deaktivieren Sie das Kontrollkästchen für die Kategorie. Sobald Sie im Auswahlfenster auf ein Audit klicken, wird dessen Beschreibung im unteren Fenster des Dialogfelds angezeigt. Die Beschreibung enthält die Komponenten, die durch das Audit überprüft werden, Beispiele für Fehler und Hinweise zum Korrigieren des Quelltext. **Optionsfenster** Der Optionssatz variiert je nach ausgewähltem Audit. Die Optionssteuerungen werden gegebenenfalls in der Beschreibung des jeweiligen Audits erläutert. Mit den Schaltflächen der Symbolleiste werden Eigenschaften in der gewünschten Form angezeigt.

Je nach Audit stehen möglicherweise noch weitere Optionen zur Verfügung. Schaltfläche **Beschreibung** Nach Kategorie Die Eigenschaften des Audits werden in erweiterbaren Gruppen angezeigt. **Alphabetisch** Die Eigenschaften des Audits werden in alphabetischer Reihenfolge angezeigt. **Schweregrad** Die Option **Schweregrad** ist für alle Audits verfügbar. Sie können jedem Audit einen Schweregrad (**Info**, **Warnung**, **Fehler** oder **Schwerwiegend**) zuweisen. Der Schwergrad gibt an, wie gravierend die Verstöße sind. Der zugewiesene Schwergrad wird im Ergebnisbericht angezeigt. Schaltflächen **Start** Das ausgewählte Audit-Set wird ausgeführt. **Abbrechen** Die Änderungen werden verworfen, und das Dialogfeld wird geschlossen. **Hilfe** Dieses

Hilfethema wird geöffnet.

Siehe auch

Überblick über Qualitätssicherungsfunktionen ( [siehe Seite 1456](#))

Audits ausführen ( [siehe Seite 269](#))

Audit-Ergebnisse anzeigen ( [siehe Seite 270](#))

2.346 QS-Metriken (Dialogfeld)

Kontextmenü▶QS-Metriken

Wählen Sie zum Öffnen des Dialogfelds **QS-Metriken** den Befehl **QS-Metriken** im Kontextmenü der Diagrammansicht, der Modellansicht oder der Klasse bzw. des Interface.

Warnung: Diese Funktion steht nur für Implementierungsprojekte zur Verfügung.

Symbolleiste	Mit den Schaltflächen in der Symbolleiste des Dialogfelds QS-Metriken werden benutzerdefinierte Metrik-Sets geladen und gespeichert sowie die Metriken festgelegt, die für die Projekte ausgeführt werden sollen. In der folgenden Tabelle sind die Schaltflächen der Symbolleiste beschrieben.
Schaltfläche	Beschreibung
Set laden	Ein Fenster zur Dateiauswahl wird geöffnet, in dem eine .mts-Datei mit einem benutzerdefinierten Metrik-Set geladen werden kann.
Set speichern unter	Ein Dialogfeld zum Speichern wird geöffnet. Geben Sie den Namen und den Speicherort für die .mts-Datei ein, in der das aktuelle Metrik-Set gespeichert werden soll.
Alles auswählen	Die Kontrollkästchen aller Metriken im Dialogfeld werden aktiviert.
Auswahl aufheben	Die Auswahl der Kontrollkästchen aller Metriken im Dialogfeld wird aufgehoben.
Standard setzen	Das Standard-Set (in der Datei default.mts) wird wiederhergestellt.
Metrik suchen	Es wird zu der Metrik gewechselt, deren Name mit dem angegebenen String beginnt.
Bereich	In diesem Feld legen Sie fest, für welche Teile des Projekts die Metriken ausgeführt werden. Klicken Sie auf den Abwärtspfeil, und wählen Sie die aktuelle Auswahl oder das gesamte Modell als Bereich. Bei der Option Auswahl werden Metriken nur für das Diagramm, den Namespace oder die Klasse ausgeführt, die Sie vor dem Öffnen des Dialogfelds Metriken ausgewählt haben. Wenn Sie Modell wählen, werden die Metriken für das gesamte Projekt ausgeführt.
Anmerkung:	Wenn in der Diagrammansicht oder der Modellansicht keine Elemente ausgewählt sind, wird die Option Bereich automatisch auf Modell gesetzt. Sollen Metriken nur für bestimmte Klassen, Namespaces oder Diagramme ausgeführt werden, müssen Sie diese bereits vor dem Öffnen des Dialogfelds Metriken auswählen.

Auswahlfenster Das Auswahlfenster enthält eine Liste der verfügbaren Metriken. Die Metriken sind nach Kategorien sortiert. Aktivieren Sie die Kontrollkästchen der Metriken, die ausgeführt werden sollen. Wenn Sie alle Metriken einer Kategorie auswählen oder ihre Auswahl aufheben möchten, aktivieren bzw. deaktivieren Sie das Kontrollkästchen für die Kategorie. Sobald Sie im Auswahlfenster auf eine Metrik klicken, wird deren Beschreibung im unteren Fenster des Dialogfelds angezeigt. Die Beschreibung enthält die Komponenten, die durch die Metrik überprüft werden, Beispiele für Fehler und Hinweise zum Korrigieren des Quelltextes.

Optionsfenster Der Optionssatz variiert je nach ausgewählter Metrik. Die Optionssteuerungen werden gegebenenfalls in der Beschreibung der jeweiligen Metrik erläutert. Mit den Schaltflächen der Symbolleiste werden Eigenschaften in der gewünschten Form angezeigt.

Je nach Metrik stehen möglicherweise noch weitere Optionen zur Verfügung. Schaltfläche **Beschreibung** Nach Kategorie Die Eigenschaften der Metrik werden in erweiterbaren Gruppen angezeigt. Alphabetisch Die Eigenschaften der Metrik werden in alphabetischer Reihenfolge angezeigt. Eigenschaftenseiten (sofern vorhanden) werden angezeigt. Aggregation Die Option **Aggregation** ist immer verfügbar. Mit ihrer Hilfe wird festgelegt, wie die Metrikergebnisse behandelt werden. Den Aggregationstyp können Sie in der Dropdown-Liste auswählen (**Summe**, **Durchschnitt**, **Maximum** usw.). Die Aggregation der Ergebnisse wird auf jeder Stufe separat durchgeführt. So werden für verschachtelte Namespaces beispielsweise die Klassen

zusammengefasst, die zu dem jeweiligen Namespace gehören. SchaltflächenStartDie ausgewählten Metriken werden ausgeführt. AbbrechenDie Änderungen werden verworfen, und das Dialogfeld wird geschlossen. HilfeDieses Hilfethema wird geöffnet.

2 Siehe auch

Überblick über Qualitätssicherungsfunktionen (siehe Seite 1456)

Metriken ausführen (siehe Seite 273)

Metrikergebnisse anzeigen (siehe Seite 274)

2.347 Sicheres Löschen (Dialogfeld)

Refactor ► Sicheres Löschen

Sie können das Dialogfeld **Sicheres Löschen** über das Hauptmenü **Refactoring** oder durch Auswahl des Befehls **Refactoring | Sicheres Löschen** im Kontextmenü bestimmter Klassendiagrammelemente öffnen. Der Befehl **Sicheres Löschen** steht für alle Quelltext generierenden Klassendiagrammelemente, nicht jedoch für Namespace-Elemente zur Verfügung.

Warnung: Diese Funktion steht nur für Implementierungsprojekte zur Verfügung.

Folgende werden gelöscht:	Elemente fehlerlos	In diesem schreibgeschützten Feld wird der Name des zu löschen Elements angezeigt.
Verwendung:		In diesem schreibgeschützten Feld werden die Verwendungen des Elements angezeigt.
Schaltflächen		
Löschen:		Das Element wird gelöscht, und das Fenster Refactoring wird geöffnet. Diese Schaltfläche steht nur zur Verfügung, wenn keine Verwendung des Elements gefunden wurde.
Verwendung anzeigen:		Diese Schaltfläche ist nur aktiv, wenn eine Verwendung des Elements gefunden wurde. Durch einen Klick auf Verwendung anzeigen öffnen Sie das Fenster Refactoring , in dem Sie die betreffenden Stellen vor dem Löschen des Elements überprüfen können.
Abbrechen:		Klicken Sie auf diese Schaltfläche, um das Dialogfeld zu schließen, ohne das Element zu löschen.

Siehe auch

Überblick zum Refactoring (siehe Seite 1456)

Sicheres Löschen von Elementen (siehe Seite 112)

2.348 Audit-Ergebnis speichern (Dialogfeld)

Fenster QS-Audits Schaltfläche Speichern

Dieses Dialogfeld wird über die Schaltfläche **Speichern** des Audit- bzw. Metrik-Ergebnisberichts geöffnet.

Anmerkung: Diese Funktion steht nur für Implementierungsprojekte zur Verfügung.

Ansicht auswählen	In diesem Listenfeld wählen Sie den Bereich der zu exportierenden Ergebnisse aus. Audit-Ergebnisse werden in Form von Registerseiten in der Ansicht mit dem Audit-Ergebnisbericht angezeigt. Das Kontextmenü des Ergebnisberichts enthält den Befehl Gruppieren nach , mit dem Sie die Ergebnisse gruppieren oder ihre Gruppierung aufheben können.
Anmerkung:	Die Option Aktive Gruppe steht nur zur Verfügung, wenn die Ergebnisse mit dem Befehl Gruppieren nach gruppiert wurden.

Folgende Ansichtsoptionen stehen zur Wahl:

Alle Ergebnisse: Wenn die Ergebnisse gruppiert sind, wird bei Auswahl dieser Option ein Bericht für alle Gruppen der aktuellen Registerseite ausgegeben. Sind die Ergebnisse nicht gruppiert, werden alle Ergebnisse für die aktuelle Registerseite exportiert.

Aktive Gruppe: Wenn die Ergebnisse gruppiert sind, können Sie eine Gruppe auf der aktuellen Registerseite auswählen. Der generierte Bericht enthält dann die Ergebnisse dieser Gruppe.

Ausgewählte Zeilen: Sie können in der Ansicht mit dem Audit-Ergebnisbericht eine oder mehrere Zeilen auswählen. Bei Auswahl von **Ausgewählte Zeilen** wird ein Bericht für diese Zeilen generiert. Format auswählenWählen Sie den Ausgabetyp im Listenfeld aus.

XML: Ein XML-basierter Bericht wird generiert.

HTML: Ein HTML-basierter Bericht wird generiert. KontrollkästchenDie Kontrollkästchen werden bei der Generierung eines HTML-Berichts aktiviert. Beschreibung hinzufügenIst diese Option aktiviert, werden die Audit-Beschreibungen in einem eigenen Ordner gespeichert und können über Hyperlinks in der Exportdatei angezeigt werden. Browser aufrufenWenn diese Option aktiviert ist, wird die erzeugte HTML-Datei im Standardanzeigeprogramm geöffnet. Ziel auswählenGeben Sie den voll qualifizierten Pfad zur Zielfile ein, oder klicken Sie auf die Schaltfläche **Durchsuchen**. SchaltflächenOKDie festgelegten Einstellungen werden übernommen, die Berichterstellung wird gestartet, und das Dialogfeld wird geschlossen. AbbrechenDie Änderungen werden verworfen, und das Dialogfeld wird geschlossen.

Siehe auch

Überblick über Qualitätssicherungsfunktionen (siehe Seite 1456)

Audits ausführen (siehe Seite 269)

Audit-Ergebnisse anzeigen (siehe Seite 270)

2.349 Verwendung suchen (Dialogfeld)

Suchen▶Verwendung suchen

In diesem Dialogfeld können Sie die Quelltextprojekte bequem nach Referenzen auf Elemente und Member sowie nach überschriebenen Elementen und Membern durchsuchen.

Option	Beschreibung
Verwendung Elements:	Referenzen auf das gewählte Element werden gesucht.
Verwendung Member:	Referenzen auf Member des gewählten Elements werden gesucht.
Abgeleitete Klassen:	Referenzen auf abgeleitete Klassen (bzw. Interfaces) werden gesucht.
Implementierungen:	Referenzen auf implementierende Klassen (Member) werden gesucht.
Überschreiben:	Referenzen auf Member, die das gewählte Element überschreiben, werden gesucht.
Using/Import einbeziehen:	Referenzen in using -/ import -Anweisungen werden gesucht.
Self überspringen:	Referenzen innerhalb des gewählten Elements werden nicht angezeigt.

Siehe auch

Quelltext nach Verwendung durchsuchen (siehe Seite 147)

2.350 Dialogfelder zur Elementauswahl

Diese Dialogfelder enthalten eine Baumstruktur mit den Inhalten, die in der Projektgruppe zur Verfügung stehen. Erweitern Sie die Projektknoten, um die verschachtelten Klassen anzuzeigen, wählen Sie das gewünschte Element aus, und klicken Sie auf **OK**.

Hierbei handelt es sich um eine Gruppe von Dialogfeldern zur Auswahl von Interaktionen, Operationen, Vorfahrklassen, instantiierten Klassen für Objekte usw. Das jeweilige Dialogfeld wird geöffnet, wenn Sie in einem Feld des Objektinspektors auf die Auswahlfläche klicken oder in den Menüknoten **Klasse auswählen** oder **Methode auswählen** den Befehl **Weitere** wählen.

Siehe auch

Instantiiieren eines Klassifizierers (siehe Seite 150)

Rollenbindung (siehe Seite 172)

2.351 Auswahlmanager

Dieses Dialogfeld gehört zu einer Gruppe von Auswahldialogfeldern, in denen Sie Elemente aus den verfügbaren Inhalten auswählen und einem bestimmten Zielbereich hinzufügen können. Alle Dialogfelder des Auswahlmanagers sind ähnlich aufgebaut, haben aber jeweils einen anderen Titel.

Dialogfeldtitel:	Der Titel des Dialogfelds variiert und entspricht dem aufrufenden Objekt oder der aufrufenden Eigenschaft.
Registerkarte Modellelemente oder Diagrammelemente :	Im Fenster auf der linken Seite des Dialogfelds wird der Inhalt angezeigt, der für das Projekt verfügbar ist. Navigieren Sie im Explorer zu einem Element, und wählen Sie es aus, um es in die Werteliste für das aufrufende Objekt zu übernehmen.
Vorhandene und/oder hinzufügbare Elemente:	In diesem Fenster werden zwei Arten von Daten angezeigt: Werte, die bereits existieren und vom aufrufenden Objekt übergeben wurden (sofern vorhanden). Werte, die aus dem Fenster auf der linken Seite übernommen wurden (sofern vorhanden).
Hinzufügen:	Ist aktiviert, wenn ein Element im linken Fenster ausgewählt wird. Wenn Sie auf die Schaltfläche klicken, wird das markierte Element der Liste im rechten Fenster hinzugefügt.
Entfernen:	Ist aktiviert, wenn ein Eintrag im rechten Fenster ausgewählt wird. Klicken Sie auf die Schaltfläche, um das Element zu entfernen. Alle entfernten Werte oder Objekte werden endgültig aus der aufrufenden Eigenschaft bzw. dem Diagramm gelöscht, wenn Sie auf OK klicken.
Alle entfernen:	Ist aktiviert, wenn Einträge im rechten Fenster vorhanden sind. Klicken Sie auf die Schaltfläche, um alle Elemente aus diesem Fenster zu entfernen. Alle entfernten Werte oder Objekte werden endgültig aus der aufrufenden Eigenschaft bzw. dem Diagramm gelöscht, wenn Sie auf OK klicken.

Siehe auch

Verknüpfungen erstellen (siehe Seite 139)

Diagramme über Hyperlinks verknüpfen (siehe Seite 125)

Modellelemente ausblenden/einblenden (siehe Seite 193)

2.352 XMI-Export (Dialogfeld)

Datei ▶ Projekt nach XMI exportieren

Dieses Dialogfeld dient zum Exportieren eines Together-Modells in eine XML-Datei, die die Modellbeschreibung in XMI enthält.

Um das Dialogfeld zu öffnen, markieren Sie in der Modellansicht den Projektstammknoten und wählen im Hauptmenü **Datei | Projekt nach XMI exportieren**. Alternativ klicken Sie in der Modellansicht mit der rechten Maustaste auf den Projektstammknoten und wählen im Kontextmenü **Projekt nach XMI exportieren**.

XMI-Typ auswählen	Wählen Sie in der Liste der unterstützten Typen den gewünschten XMI-Typ aus. Einige der von Together unterstützten Elemente sind in IBM Rational Rose nicht zulässig. Mit Hilfe der Option XMI für UML 1.3 (Unisys-Erweiterung, empfohlen für Rose) können alle entsprechenden Elemente entfernt werden. Für erkannte Meldungen werden auf der Registerkarte Ausgabe entsprechende Meldungen angezeigt.
XMI-Codierung	Klicken Sie auf den Abwärtspfeil, und wählen Sie die gewünschte Codierung des Ausgabe-Streams aus.
Exportziel auswählen:	Geben Sie den voll qualifizierten Namen der resultierenden XML-Datei ein. Klicken Sie gegebenenfalls auf die Schaltfläche Durchsuchen , um die Zielposition anzugeben.
Schaltflächen	
Exportieren	Die Eingaben werden übernommen, und das Modell wird in XMI exportiert.
Abbrechen	Ihre Eingaben werden verworfen, das Dialogfeld wird geschlossen, und das Modell wird nicht exportiert.

Siehe auch

Überblick zum Importieren und Exportieren (siehe Seite 1459)

2.353 XMI-Import (Dialogfeld)

Datei | Projekt aus XMI importieren

Dieses Dialogfeld dient zum Importieren einer XML-Datei, die die Modellbeschreibung in XMI enthält.

Um das Dialogfeld zu öffnen, markieren Sie in der Modellansicht den Projektstammknoten und wählen im Hauptmenü **Datei | Projekt aus XMI importieren**. Alternativ klicken Sie in der Modellansicht mit der rechten Maustaste auf den Projektstammknoten und wählen im Kontextmenü **Projekt aus XMI importieren**.

Quelldatei auswählen	Geben Sie den voll qualifizierten Namen der XML-Datei ein. Klicken Sie gegebenenfalls auf die Schaltfläche Durchsuchen , und navigieren Sie zur Datei.
Schaltflächen	
Importieren	Die XML-Datei wird importiert.
Abbrechen	Ihre Eingaben werden verworfen, das Dialogfeld wird geschlossen, und das Modell wird nicht importiert.

Siehe auch

Überblick zum Importieren und Exportieren (siehe Seite 1459)

2.354 CodeGuard-Konfiguration

Tools ▶ CodeGuard-Konfiguration

Verwenden Sie das Dialogfeld CodeGuard-Konfiguration, um festzulegen, wie sich der CodeGuard-Laufzeit-Debugger verhält.

Anmerkung: CodeGuard steht nur für C++-Projekte zur Auswahl.

Element	Beschreibung
Aktivieren (CodeGuard)	Aktiviert oder deaktiviert CodeGuard.
Stack-Füllfrequenz	Gibt an, wie häufig CodeGuard den nicht initialisierten Teil des Laufzeit-Stack mit einem einmaligen Byte-Muster füllt.
Wert	Frequenz
-1	Nie
0	Nach jedem Aufruf einer Laufzeitfunktion, die von CodeGuard abgedeckt wird.
n [0...15]	Nach allen 2^n Aufrufen einer Laufzeitfunktion, die von CodeGuard abgedeckt wird. Zum Beispiel, wenn n 1 ist, dann wird der Stack jedes zweite Mal gefüllt, wenn eine Laufzeitfunktion aufgerufen wird.
Statistik	Gibt eine Funktions- und Ressourcenverwendungsstatistik aus.
Ressourcen-Lecks	Gibt Ressourcen-Lecks an, die nach dem Ende der Anwendung entdeckt wurden.
An OutputDebugString senden	Verwendet die Funktion <code>OutputDebugString</code> dazu, CodeGuard-Meldungen an den externen Debugger zu senden.
An Protokoll anhängen	Hängt das Fehlerprotokoll an das vorhandene Protokoll an. Wenn diese Option deaktiviert ist, überschreibt CodeGuard das vorhandene Fehlerprotokoll.
Wiederkehrende Fehler	Gibt Fehler an, die wiederholt in einer Funktion auftreten.
Anzahl von Fehlermeldungen begrenzen	Beschränkt die Anzahl der berichteten Fehler. Sie können einen Maximalwert von 65535 eingeben.
Aktivieren (Fehlermeldung)	Aktiviert ein Fehlermeldungsfeld. Wenn Sie eine CodeGuard-aktivierte Anwendung außerhalb von RAD Studio ausführen, wird das Fehlermeldungsfeld angezeigt, wenn Laufzeitfehler auftreten.
Titel	Legt den Text für die Titelleiste des Dialogfelds fest.
Meldung	Legt die angezeigte Fehlermeldung fest.
Debug-Info lesen	Aktiviert CodeGuard, um die Debug-Informationen in Ihrem Projekt dazu zu verwenden, auf eine Quelltextzeile zu verweisen, wenn ein Laufzeitfehler aufgetreten ist.
Quellpfad	Wenn der Quellcode sich an einem anderen Speicherort befindet als die EXE-Datei, geben Sie hier den Pfad an (oder Pfade, getrennt durch Semikolon). CodeGuard überprüft zunächst den eigenen Quellpfad und dann den IDE-Debug-Quellpfad (sofern es in der IDE ausgeführt wird).

Ressourcenoptionen

Verwenden Sie die Seite Ressourcenoptionen, um anzugeben, wie CodeGuard die verschiedenen Ressourcentypen behandelt.

Element	Beschreibung
Ressourcen	Listet die Ressourcentypen auf, die CodeGuard behandeln kann.
Ressourcen	Beschreibung
memoryblock	Der Speicher, der von den Funktionen <code>malloc</code> und <code>free</code> verwaltet wird.
object	Speicher, der von den Operatoren <code>new</code> und <code>delete</code> verwaltet wird.
objectarray	Speicher, der von den Operatoren <code>new[]</code> und <code>delete[]</code> verwaltet wird.
file handle	Datei, die von den Funktionen <code>open</code> und <code>close</code> verwaltet wird.
file stream	Datei, die von den Funktionen <code>fopen</code> und <code>fclose</code> verwaltet wird.
pipe stream	Ein Befehls-Prozessor-Pipe, verwaltet von den Funktionen <code>_popen</code> und <code>_pclose</code> .
directory stream	Verzeichnis, das von den Funktionen <code>opendir</code> und <code>closedir</code> verwaltet wird.
Verfolgung aktivieren	Aktiviert die Verfolgung der ausgewählten Ressource. Eine deaktivierte Verfolgung führt zu geringerer Speicherbeanspruchung und schnellerer Ausführung.
Ressourcen-Lecks verfolgen	Gibt Ressourcen-Zuweisungen an, die keine entsprechende Freigabe haben. Zum Beispiel: ein Leck kann durch fehlende Freigabe eines Datei-Handles vor dem Programmabbruch entstehen.
Meldung bei ungültigen Handle- oder Ressourcen-Parametern	Gibt eine nicht korrekte Verwendung der Ressourcen in Funktionsargumenten an.
Freigabe verzögern	<p>Verfolgt die ausgewählte Ressource nach der Freigabe. Ist die Option Freigabe verzögern aktiv, kennzeichnet CodeGuard jede Ressource einmal, sobald sie freigegeben ist. Gleichzeitig werden Windows und die Laufzeit-Bibliotheken daran gehindert, die Ressource wieder zu verwenden.</p> <p>Einige Ressourcen, z.B. die Stack-Speicherzuweisungen lassen sich nicht in die Warteschlange für die verzögerte Freigabe einreihen.</p>
Element	Beschreibung
Länge der Verzögerungswarteschlange	Gibt die Anzahl der Objekte an, die für die verzögerte Freigabe in die Warteschlange eingereiht werden können. Sie können einen Maximalwert von 65535 Objekten eingeben.
Max. Größe für Speicherblöcke	Gibt die maximale Anzahl der Speicherblöcke an, die CodeGuard in der Warteschlange speichern kann. Sie können einen Maximalwert von 65535 Bytes eingeben.

Funktionsoptionen

Verwenden Sie die Seite Funktionsoptionen, um anzugeben, wie CodeGuard die verschiedenen Funktionstypen behandelt.

Element	Beschreibung
Funktionen	Listet die Funktionen auf, die CodeGuard verfolgen kann.
Funktionsverfolgung deaktivieren	Deaktiviert die Funktionsverfolgung für die ausgewählten Funktionen.
Speicherzugriffsfehler	Gibt einen Laufzeitfehler an, wenn eine Funktion einen Zeiger auf ungültigen Speicher verwendet.
Jeden Aufruf protokollieren	Gibt jeden Aufruf der ausgewählten Funktionen an.
Warnungen	<p>Gibt Situationen an, in denen die Anwendung auf einen Speicherplatz jenseits der Maximalgröße des Puffers zugreift.</p> <p>Warnungen werden nur für die folgenden Laufzeit-Bibliotheksfunktionen ausgegeben: <code>strcmp</code>, <code>strnicmp</code>, <code>strncpy</code>, <code>_fstrcmp</code>, <code>_fstrnicmp</code>, <code>memcmp</code>, <code>memicmp</code>, <code>_fmemcmp</code>, <code>_fmemicmp</code>, <code>fnmerge</code>, <code>fnsplit</code>, <code>getcurdir</code>.</p>

Fehler Funktionsergebnissen	bei	Gibt an, wenn die ausgewählte Funktion einen Wert zurückgibt, der einen Fehler anzeigt.
Ungültige Handle- oder Ressourcen-Parameter		Wenn es sich bei einem der ausgewählten Funktionsparameter um einen Handle- oder Ressourcen-Bezeichner handelt, sollte überprüft werden, ob dieser korrekt zugewiesen und aktuell gültig ist.
Standard-Funktionsoptionen setzen		Zeigt das Dialogfeld Standard-Funktionsoptionen an, das Sie dazu verwenden können, die Standard-Funktionsoptionen anzuzeigen und zu setzen.
Standard-Funktionsoptionen		Weist den ausgewählten Funktionen die Standard-Funktionsoptionen zu.

Ignorierte Module

Auf der Seite Ignorierte Module geben Sie jene Module an, die CodeGuard beim Fehlerbericht überspringen soll.

Siehe auch

[Überblick zu CodeGuard](#)

[CodeGuard verwenden](#)

2.355 Tools-Optionen

Tools ▶ Tools konfigurieren

Verwenden Sie dieses Dialogfeld, um festzustellen, welche Programme im Menü Tools zur Verfügung stehen.

Element	Beschreibung
Tools	Listet die Programme auf, die dem Menü Tools hinzugefügt wurden.
Hinzufügen	Zeigt das Dialogfeld Tool-Eigenschaften an, in dem Sie ein Programm hinzufügen können.
Löschen	Löscht das aktuell in der Liste Tools ausgewählte Programm.
Bearbeiten	Zeigt das Dialogfeld Tool-Eigenschaften an.
Pfeilsymbole	Verwenden Sie die Pfeilschaltflächen, um die Programme in der Liste Tools neu anzurichten. Die Programme werden im Menü Tools in der gleichen Reihenfolge aufgeführt.

Tip: Hinzugefügte Programme werden im unteren Bereich des Menüs Tools angezeigt.

2.356 Objekt-Info bearbeiten

Tools ▶ **Objektablage** ▶ **Schaltfläche Bearbeiten**

Verwenden Sie dieses Dialogfeld, um die Informationen über ein Objekt in der Objektablage zu bearbeiten.

Element	Beschreibung
Kategorie	Zeigt die Kategorien an, die im Dialogfeld Objektgalerie erscheinen, wenn Sie Datei ▶ Neu ▶ Weitere wählen.
Titel	Gibt den Titel des ausgewählten Elements an.
Beschreibung	Gibt die Beschreibung des ausgewählten Elements an. Die Beschreibung wird angezeigt, wenn Sie im Dialogfeld Objektgalerie mit der rechten Maustaste klicken und Details anzeigen wählen.
Autor	Gibt den Namen des Autors des ausgewählten Elements an.
Durchsuchen	Zeigt das Dialogfeld Symbol wählen an, in dem Sie ein anderes Symbol zur Darstellung des Objekts im Dialogfeld Objektgalerie auswählen können. Sie können Bitmaps beliebiger Größe verwenden, die jedoch immer im Format 60 x 40 Pixel angezeigt werden.
Neue Kategorie	Zeigt das Dialogfeld Neuer Kategorienname an, in dem Sie den Namen für eine neue Kategorie in der Objektablage eingeben können.

2.357 Bearbeitungs-Tool

Tools > Build-Tools > Schaltfläche Hinzufügen oder Bearbeiten

Verwenden Sie dieses Dialogfeld zum Hinzufügen oder Ändern von Build-Tool-Überschriften oder Dateizuordnungen.

Element	Beschreibung
Titel	Geben Sie einen Namen für das Tool ein.
Standarderweiterungen	Führen Sie die Dateierweiterungen auf, die das Tool standardmäßig compilieren soll. Das Tool wird automatisch für alle Dateien mit dieser Erweiterung in Ihrem Projekt ausgeführt, wenn Sie das Projekt compilieren oder erstellen. Trennen Sie mehrere Erweiterungen durch Semikolon. In der Projektverwaltung können Sie die Dateien mit dieser Erweiterung mit der rechten Maustaste anklicken, um auf das Tool zuzugreifen.
Andere Erweiterungen	Führen Sie die Dateierweiterungen auf, für die das Tool bei Bedarf verwendet werden soll. Dies ist für Präprozessoren, Archivierungs-Tools, Diagnose-Tools oder automatische Hilfegeneratoren nützlich. Trennen Sie mehrere Erweiterungen durch Semikolon. In der Projektverwaltung können Sie die Dateien mit dieser Erweiterung mit der rechten Maustaste anklicken, um auf das Tool zuzugreifen.
Zielerweiterung	Geben Sie die Dateierweiterung an, die das Tool erstellen soll.
Befehlszeile	Geben Sie die Befehlszeile ein, die beim Erstellen einer Datei dieses Typs ausgeführt werden soll.
Filter	Geben Sie einen benutzerdefinierten Filter in einem Package (mit der Tools-API erstellt) an, mit dem die Ausgabeinformationen des Tools gefiltert und im Fenster Meldungen angezeigt werden sollen. Ist kein Filter angegeben, wird ein Standardfilter benutzt. Der Standardfilter sendet alle Ausgaben des Tools an stdout und Fehler (stderr) an das Fenster Meldungen.
Makro	Zeigt eine Liste der Makros an, die Sie in der Befehlszeile verwenden können (z.B. ruft \$NAME einen Dateinamen ab). Die Makros werden erweitert, wenn das Tool ausgeführt wird. Doppelklicken Sie auf das Makro oder klicken Sie Einfügen an, um es in die Befehlszeile einzufügen.
Einfügen	Fügt das markierte Makro in die Befehlszeile ein.

2.358 Visual Studio-Projekt exportieren

Tools > Nach Visual Studio exportieren

Verwenden Sie dieses Dialogfeld, um das aktuelle Projekt in ein Visual Studio-Projekt zu konvertieren.

Element	Beschreibung
Name	Enthält den Namen des aktuellen Projekts. Sie können diesen Namen ändern.

2.359 Versionsverwaltung

In der Ansicht Versionsverwaltung können Sie frühere Versionen einer Datei anzeigen und miteinander vergleichen. In dieser Ansicht lassen sich z.B. verschiedene Sicherungsversionen, gespeicherte lokale Änderungen oder der Puffer mit nicht gespeicherten Änderungen für die aktive Datei anzeigen. Unterliegt die aktuelle Datei der Versionskontrolle, sind alle Revisionstypen in der Versionsverwaltung zu sehen.

Die Versionsverwaltung wird auf der Registerkarte Historie angezeigt, die sich in der Mitte der IDE rechts von der Registerkarte Code befindet. Die Versionsverwaltung enthält die folgenden Registerkarten:

Seite	Beschreibung
Inhalt	Zeigt die aktuelle Version und frühere Versionen der Datei an.
Info	Zeigt alle Labels und Kommentare für die aktive Datei an.
Unterschied	Zeigt die Unterschiede zwischen den ausgewählten Versionen der Datei an.

Schaltflächen in der Symbolleiste der Versionsverwaltung

Schaltfläche	Beschreibung
	Revisionsinfo aktualisieren aktualisiert die Revisionsliste mit den nicht gespeicherten Änderungen, die an der Datei vorgenommen wurden.
	Auf vorherige Revision zurücksetzen macht die ausgewählte Version zur aktuellen Version (nur auf den Seiten Inhalt und Info verfügbar). Durch das Zurücksetzen auf eine vorherige Version gehen nicht gespeicherte Änderungen im Editorpuffer verloren.
	Blättern synchronisieren synchronisiert den Bildlauf auf den Seiten Inhalt und Unterschied mit dem im Quelltext-Editor. Dabei wird jeweils die Textzeile, die den Cursor enthält, mit der nächsten passenden Textzeile in der anderen Ansicht synchronisiert. Sind im betreffenden Dateiabschnitt keine übereinstimmenden Zeilen vorhanden, erfolgt die Synchronisierung nach den Zeilennummern.
	Zum nächsten Unterschied wechseln positioniert die Quellansicht auf der Seite Unterschied auf den nächsten Block mit abweichendem Code.
	Zum vorherigen Unterschied wechseln positioniert die Quellansicht auf der Seite Unterschied auf den vorherigen Block mit abweichendem Code.
	Textbewegungen folgen wechselt zur gleichen Zeile in der Quellansicht, wenn zwischen den Ansichten umgeschaltet wird.

Mit den folgenden Symbolen werden Dateiversionen in den Revisionslisten dargestellt.

Revisionssymbole in der Versionsverwaltung

Symbol	Beschreibung
	Die zuletzt gespeicherte Dateiversion.
	Eine Sicherungsdatei-Version.
	Die Dateiversion, die sich im Puffer befindet und nicht gespeicherte Änderungen enthält.
	Eine Dateiversion, die in einer Ablage der Versionskontrolle gespeichert ist.



Eine Dateiversion, die aus einer Ablage der Versionskontrolle ausgecheckt wurde.

2 Siehe auch

Ein erster Bilck auf die IDE ([siehe Seite 1363](#))

Mit der Versionsverwaltung arbeiten ([siehe Seite 52](#))

2.360 Objektablage

Tools ▶ Objektablage

Verwenden Sie dieses Dialogfeld zum Bearbeiten, Verlagern und Entfernen von Projekt-Templates.

Element	Beschreibung
Kategorien	Listet die verfügbaren Kategorien auf, die in der Objektablage Projekt- und Formular-Templates enthalten.
Objekte in der Objektablage	Listet die in den Kategorien vorhandenen Projekt- und Formular-Templates auf.
Bearbeiten	Zeigt das Dialogfeld Objekt-Info bearbeiten an, in dem Sie die Eigenschaften der Templates in der Objektablage bearbeiten können.
Löschen	Entfernt eine Template aus der Objektablage.

2.361 Bereich für Exception hinzufügen

Tools▶**Optionen**▶**Debugger-Optionen**▶**Native BS-Exceptions**▶**Schaltfläche Hinzufügen**

Verwenden Sie dieses Dialogfeld zum Festlegen des Bereichs von Exceptions, für die das Programm die Ausführung anhalten soll. Der numerische Wert, den Sie einer Exception zuordnen, wird unten auf der Liste Exceptions auf der Seite Native BS-Exceptions des Dialogfeldes Debugger-Optionen angezeigt.

Element	Beschreibung
Untergrenze	Geben Sie den unteren Wert für den Bereich ein.
Obergrenze	Geben Sie den oberen Wert für den Bereich ein.

Tip: Wenn Sie in den Feldern Untergrenze und Obergrenze Werte angeben, wird die Ausführung bei jeder im angegebenen Bereich liegenden Exception angehalten.

2.362 Sprach-Exception hinzufügen

Tools▸**Optionen**▸**Debugger-Optionen**▸**Borland .NET Debugger**▸**Sprach-Exceptions**▸**Schaltfläche Hinzufügen**

Verwenden Sie dieses Dialogfeld zum Hinzufügen einer Sprach-Exception zu der Liste auf der Seite Sprach-Exceptions. Zum Einfügen der Schaltfläche Bei Sprach-Exceptions benachrichtigen in eine Symbolleiste verwenden Sie **Ansicht**▸**Symbolleisten**▸**Anpassen**.

Element	Beschreibung
Textfeld	Geben Sie einen beliebigen intrinsischen, framework-definierten oder benutzerdefinierten Sprachtyp, z.B. <code>System.Windows.Forms.Panel</code> , <code>Project123.WinForm</code> oder <code>MyClass</code> ein.

2.363 Aktualisierung durchführen

Verwenden Sie dieses Dialogfeld, um vorgeschlagene Quelltext-Änderungen zu überprüfen, wenn Sie die Typbibliothek im Typbibliotheks-Editor aktualisieren, speichern oder registrieren möchten.

Element	Beschreibung
Aktualisierungen auswählen	Zeigt die Änderungen in der Reihenfolge an, in der sie an Ihrem Projekt vorgenommen werden. Markieren Sie das Kontrollkästchen neben jeder Änderung oder heben Sie dessen Markierung auf, um Änderungen an der betreffenden Datei zu bestätigen bzw. zu unterbinden. Wenn Sie die Auswahl einer Änderung aufheben, von der nachfolgende Änderungen abhängen (z.B. die Erstellung einer Datei, in die nachfolgend Code eingefügt wird), dann werden die nachfolgenden Änderungen automatisch deaktiviert.
Details	Zeigt alle Änderungen an, die zur Durchführung der aktuell ausgewählten Änderung vorgenommen werden. Wenn Sie auf OK klicken, werden die hier vorgenommenen Änderungen, einschließlich der innerhalb dieses Dialogfeldes vorgenommenen Änderungen, für jede in der Liste Aktualisierungen auswählen markierte Aktualisierung übernommen. Besteht eine Aktualisierung aus neuem Quelltext, der einer Datei hinzugefügt wird, dann enthält das Feld Details ein Bearbeitungsfeld, in dem der neue Quelltext angezeigt wird. Wird durch eine Aktualisierung vorhandener Quelltext geändert, dann enthält die Registerkarte Details zwei Textfenster: das erste zeigt den neuen geänderten Quelltext und das zweite zeigt den Quelltext in seiner ursprünglichen Fassung.
Dieses Dialogfeld nicht wieder anzeigen	Legt fest, ob dieses Dialogfeld jedes Mal angezeigt wird, wenn Sie eine Typbibliothek verändern und dann zu aktualisieren, zu speichern oder zu registrieren versuchen. Markieren Sie diese Option, wenn Änderungen ohne erneute Überprüfung durch Sie übernommen werden sollen. Durch die Auswahl dieser Option wird die Option Änderungen vor Aktualisierung anzeigen auf der Registerkarte Typbibliothek des Dialogfeldes Tools > Optionen > Umgebungsoptionen > Delphi-Optionen deaktiviert.

2.364 ASP.NET

Tools ▶ Optionen ▶ HTML/ASP.NET-Optionen ▶ ASP. NET

Verwenden Sie dieses Dialogfeld, um Standardinformationen für das Erstellen von ASP.NET-Webanwendungen festzulegen.

Element	Beschreibung
Name	Gibt den Internet-Browser an, mit dem die Webanwendung in der IDE geöffnet wird.
Path	Gibt den Pfad zu der ausführbaren Datei des Internet-Browsers an. Um den Pfad zu ändern, wählen Sie aus der Dropdown-Liste Name Andere und suchen dann mithilfe der Ellipsen-Schaltfläche die ausführbare Datei eines Browsers.
Parameter	Optional. Geben Sie die Parameter ein, die an die Internet-Browser-Anwendung übergeben werden sollen.
Path	Gibt den Pfad zu der ausführbaren Datei des Cassini Web Server an.
Port	Gibt den vom Cassini Web Server verwendeten TCP/IP-Port an.
Basisverzeichnis	Legt den Standardverzeichnispfad für neue Webanwendungen fest. Dieser Pfad wird im Dialogfeld Neue ASP.NET-Anwendung angezeigt. Sie können den Pfad aber bei Bedarf ändern.
Webserver	Legen Sie den Standard-Webserver für neue Webanwendungen fest. Beim Erstellen einer neuen Webserver-Anwendung wird dieser Server im Dialogfeld Neue ASP .NET-Anwendung angezeigt. Sie können den Server aber bei Bedarf ändern.

Anmerkung: Sie können den Cassini Web Server von <http://www.asp.net/Projects/Cassini/Download> herunterladen.

2.365 CodeGear-Debugger

Tools > Optionen > Debugger-Optionen > CodeGear-Debugger

Verwenden Sie diese Seite zum Festlegen von Debugger-Optionen für die IDE.

Element	Beschreibung
Seiteneffekte in neuen Ausdrücken	Bewirkt die Überwachung eines Ausdrucks, auch wenn dadurch Seiteneffekte auftreten können. Diese Option kann für einzelne überwachte Ausdrücke im Dialogfeld Darstellung überwachter Ausdrücke gesetzt werden. Standardmäßig ist die Option deaktiviert.
Mehrfach-Evaluatoren	Gibt an, dass der entsprechende Evaluator (C++ oder Delphi) für jedes Modul verwendet wird, das in den Prozess geladen wurde, der gedebuggt wird. Angenommen, Ihr Delphi-Programm lädt ein DLL-Build mit der Personality C++, so wird der C++-Evaluator verwendet, wenn der Debugger auf die C++-DLL stößt. Ist diese Option deaktiviert, wird nur der für die aktive Personality geeignete Evaluator verwendet. Beachten Sie, dass diese Option nur für den CodeGear Win32 Debugger verfügbar ist.
Debug Spawned-Prozessen	Überprüft Prozesse, die von dem gerade überprüften Prozess erzeugt werden. Ist diese Option nicht aktiviert, werden diese Prozesse zwar ausgeführt, aber vom Debugger nicht untersucht.
Nicht-Benutzer-Haltepunkte ignorieren	Stoppt nur an Haltepunkten, die Sie explizit in der IDE gesetzt haben. Wenn diese Option markiert ist, ignoriert der native Debugger hart codierte <code>int 3</code> -Haltepunkte genauso wie Haltepunkte, die aus einem Aufruf der Windows API-Methode <code>DebugBreak</code> resultieren. Außerdem wird eine native Anwendung, die im Debug-Modus ausgeführt wird, auch nicht durch Drücken von F12 angehalten, wenn diese Option markiert ist. Der verwaltete Debugger ignoriert auch Haltepunkte, die aus einem Aufruf von <code>System.Diagnostics.Debugger.Break</code> resultieren. Das Ändern dieser Option wirkt sich sofort aus. Der Vorgabewert ist Aus.
Vererbung anzeigen	Schaltet das Fenster des Debug-Inspektors in den Ausschnitten Daten, Methoden und Eigenschaften zwischen zwei Modi um: einem, der alle intrinsischen und vererbten Datenelemente oder Eigenschaften einer Klasse anzeigt und einem, der nur die in der Klasse deklarierten Elemente anzeigt. Bei Klassenobjekten können Sie damit festlegen, ob Elemente, die Bestandteil einer Vorfahrklasse sind, oder nur Elemente, die in der aktuellen Klasse deklariert sind, angezeigt werden.
Voll qualifizierte Namen anzeigen	Ermöglicht die Anzeige der geerbten Elemente mit ihren voll qualifizierten Namen.
Nach Namen sortieren	Sortiert die Seiten im Debug-Inspektor alphabetisch. Standardmäßig werden die Seiten entsprechend der Deklarationsreihenfolge sortiert. Beachten Sie, dass diese Option in Delphi für Win32-Projekten ignoriert wird.
Inspektor immer im Vordergrund	Zeigt alle Debugger-Fenster im Vordergrund an, auch wenn sie nicht aktiv sind.
Im Editor eingebettet	Legt fest, dass die Disassemblierungsansicht als integrierter Bestandteil des CPU-Fensters angezeigt wird. Dies ist die Standardeinstellung.
Andockbares Fenster trennen	Legt fest, dass die Disassemblierungsansicht als eigenes Fenster, das in der IDE verschoben werden kann, angezeigt wird.
Suchpfad Debug-Symbole für	Legt den Pfad zu den Debug-Symboldateien (<code>.pdb</code>) und den <code>.tds</code> -Dateien fest. Diese Dateien werden normalerweise zusammen mit der ausführbaren Datei oder der dynamischen Link-Bibliothek (DLL) gespeichert.

Pfad für Debugger	<p>Legt die Verzeichnisse fest, in denen die CodeGear-Debugger nach Unit-Dateien suchen, die im Projektsuchpfad oder im Projektquellpfad nicht gefunden werden.</p> <p>Die zusätzlichen Verzeichnisse werden in der folgenden Reihenfolge durchsucht:</p> <ol style="list-style-type: none">1. Projektspezifischer Pfad für Debugger, der auf der Seite Projekt>Optionen>Debugger angegeben wurde.2. Suchpfad, der folgendermaßen festgelegt ist: Für Delphi für Win32: auf der Seite Tools>Optionen>Umgebungsoptionen>Delphi-Optionen>Bibliothek — Win32. Für Delphi.NET auf der Seite Tools>Optionen>Umgebungsoptionen>Delphi-Optionen>Bibliothek — NET. Für C++ auf der Seite Tools>Optionen>Umgebungsoptionen>C++-Optionen>Pfade und Verzeichnisse.3. Pfad für Debugger (diese Option) für Projekte, die über keinen projektspezifischen Pfad für Debugger verfügen und beim Debuggen ohne ein geladenes Projekt. <p>Wenn kein Projekt in der IDE geladen ist, werden nur die Verzeichnisse durchsucht, die in dieser Option angegeben sind.</p>
-------------------	--

Siehe auch

Die Suchreihenfolge für Debug-Symboltabellen festlegen (siehe Seite 36)

2.366 Code Insight

Tools > Optionen > Editor-Optionen > Code Insight

Verwenden Sie diese Seite zum Konfigurieren der Funktionen von Code Insight für die Bearbeitung von Quelltext im Quelltext-Editor.

Anmerkung: HTML und CSS unterstützen nur die Features Programmierhilfe, Fehlermarkierung und Quelltext-Template-Vervollständigung.

Element	Beschreibung	
Typ der Quelldatei	Zeigt eine Liste der Programmiersprachen an, für die sich die Funktionen von Code Insight verwenden lassen. Pro Sprache lassen sich verschiedene Code Insight-Optionen angeben.	
Programmierhilfe	Zeigt eine Liste mit Eigenschaften, Methoden und Ereignissen an, wenn Sie im Quelltext-Editor einen Klassennamen gefolgt von einem Punkt eingeben. Sie können dann ein Element auswählen und mit ENTER in Ihren Quelltext einfügen. Wenn diese Option nicht aktiviert ist, können Sie die Programmierhilfe mit STRG+LEER aufrufen. Diese Option ist standardmäßig aktiviert.	
Code-Parameter	Zeigt die Argumente eines Methodenaufrufs bei dessen Eingabe in den Quelltext an. Wenn diese Option nicht aktiviert ist, können Sie die Code-Parameter mit UMSCHALT+STRG+LEER aufrufen. Diese Option ist standardmäßig aktiviert.	
Auswertung durch Kurzhinweis	Zeigt den aktuellen Wert einer Variable an, wenn Sie den Cursor darüber positionieren. Diese Funktion ist verfügbar, wenn die Programmausführung während einer Debug-Sitzung pausiert. Diese Option ist standardmäßig aktiviert.	
Symbolinformation durch Kurzhinweis	Zeigt Deklarationsinformationen für beliebige Bezeichner in einem Hinweisenster an, wenn die Maus im Quelltext-Editor darauf gesetzt wird. Diese Option ist standardmäßig aktiviert.	
Symbolbeschreibung	Zeigt eine Kurzbeschreibung in einem Kurzhinweisenster an, wenn der Cursor über einem Symbol im Quelltext-Editor steht. Das Fenster enthält auch Links zu weiteren Informationen (falls verfügbar). Diese Option ist standardmäßig aktiviert.	
Fehlermarkierung	Unterstreicht ungültigen Code und HTML in roter Farbe. Wenn Sie den Mauszeiger über den ungültigen Text bewegen, wird ein Hinweisenster angezeigt, das die mögliche Fehlerursache beschreibt. Diese Option ist standardmäßig aktiviert.	
Blockvervollständigung	Der Editor fügt das Schließen-Symbol für einen Block automatisch ein, wenn Sie den Block beginnen und ENTER drücken. Diese Option ist standardmäßig aktiviert. Im Drop-Down-Menü Blockvervollständigung legen Sie das Verhalten der Blockvervollständigung fest, wenn Sie vorhandene Anweisungen mit Blocksymbolen versehen.	
Mit Zeilenumbruch	Setzt den Cursor in den Block, den Sie soeben erstellt haben.	
Ohne Zeilenumbruch	Setzt den Cursor hinter den Block, den Sie soeben erstellt haben.	
Nur neue Blöcke	Ruft die Blockvervollständigung nur auf, wenn Sie einen neuen Block beginnen.	
Programmierhilfe: automatisch	Klammern	Fügt automatisch Klammern in Funktionsaufrufe ein, wenn Sie die Programmierhilfe zulassen. Diese Option ist standardmäßig aktiviert.

Quelltext-Template-Vervollständigung		Fügt automatisch eine Quelltext-Template ein, wenn Sie einen Bezeichner eingeben, der dem Namen einer Template entspricht und TAB drücken. Diese Option ist standardmäßig aktiviert.
Templates vervollständigen	automatisch	Ruft die Quelltext-Template-Vervollständigung auf, wenn Sie nach der Eingabe des Namens einer vorhandenen Quelltext-Template die Leertaste drücken. Wenn diese Option nicht aktiv ist, müssen Sie TAB drücken, um die Template-Vervollständigung aufzurufen, nachdem Sie den Template-Namen eingegeben haben. Diese Option ist standardmäßig aktiviert.
Template-Hinweise		Aktiviert die Template-Hinweise. Template-Hinweise erscheinen, wenn Sie eine Template in den Quelltext-Editor einfügen und zwischen den vordefinierten Cursorpositionen in der Template mit der Taste TAB blättern. Diese Option ist standardmäßig deaktiviert.
Verzögerung		Legt die Zeitspanne fest, die vergeht, bevor ein Code Insight-Fenster angezeigt wird. Sie können Niedrig, Mittel und Hoch auswählen.

Siehe auch

Überblick zum Quelltext-Editor (siehe Seite 1373)

Code Insight verwenden (siehe Seite 48)

Klassenvervollständigung verwenden (siehe Seite 47)

2.367 Farben

[Tools](#)▶[Optionen](#)▶[Editor-Optionen](#)▶[Programmierhilfe](#)▶[Farben](#)

Auf der Seite Farben können Sie das Aussehen des Fenster Programmierhilfe ändern.

Element	Beschreibung
Quelldateityp	Zeigt eine Liste der Programmiersprachen an, für die sich die Funktionen der Programmierhilfe verwenden lassen. Pro Sprache lassen sich verschiedene Programmierhilfe-Optionen angeben.
Farben für die Listenfelder der Programmierhilfe	Gibt die Farbe für jede Komponente im Fenster Programmierhilfe an.

Siehe auch

Überblick zum Quelltext-Editor (↗ siehe Seite 1373)

Programmierhilfe verwenden (↗ siehe Seite 48)

Klassenvervollständigung verwenden (↗ siehe Seite 47)

2.368 Quelltextvorlagen

Tools ▶ **Optionen** ▶ **Editor-Optionen** ▶ **Quelloptionen**

Verwenden Sie diese Seite zum Verwalten von Quelltext-Templates. Quelltextvorlagen enthalten häufig verwendete Programmieranweisungen (wie if-, while- und for-Anweisungen), die Sie in Ihren Quelltext einfügen können.

Element	Beschreibung
Vorlagen	Listet die Namen und Beschreibungen von Quelltext-Templates auf.
Hinzufügen	Zeigt das Dialogfeld Template hinzufügen an, in dem Sie einen Namen und eine Beschreibung für eine neue Template eingeben können.
Bearbeiten	Zeigt das Dialogfeld Template bearbeiten für die in dem Feld Templates markierte Template an.
Löschen	Löscht die in dem Feld Templates markierte Template.
Code	Zeigt die Programmieranweisungen für die markierte Quelltext-Template an. Beim Hinzufügen einer Quelltext-Template geben Sie die Programmieranweisungen in das Feld Code ein.
Import	Zeigt das Dialogfeld Quelltext-Template-Datei öffnen an, in dem Sie zu einer .dci-Datei navigieren können. .dci-Dateien enthalten Quelltext-Template-Sammlungen für einen bestimmten Quelltextdateityp.
Export	Zeigt das Dialogfeld Templates exportieren als an, in dem Sie die aktuell markierte Template als .dci-Datei speichern können.

Tip: Im Quelltext-Editor können Sie mit der Tastenkombination STRG+J eine Liste der verfügbaren Quelltext-Templates einblenden.

2.369 Farbe

Tools▶**Optionen**▶**Editor-Optionen**▶**Farbe**. Verwenden Sie diese Seite zum Festlegen der Farben für die verschiedenen Elemente Ihres Quelltextes im Quelltext-Editor.

Element	Beschreibung
Farb-Schnelleinstellung	Erlaubt das schnelle Konfigurieren der Anzeige des Quelltext-Editors mit vordefinierten Farbkombinationen. Wählen Sie eine Farb-Schnelleinstellung aus der Dropdown-Liste aus. Der Beispiel-Quelltext zeigt, wie sich Ihre Einstellungen im Quelltext-Editor auswirken.
Fett	Formatiert das ausgewählte Element in Fettschrift.
Kursiv	Formatiert das ausgewählte Element in Kursivschrift.
Unterstrichen	Stellt das ausgewählte Element unterstrichen dar.
Vordergrund	Zeigt das ausgewählte Quelltextelement in der Standardsystemfarbe für den Vordergrund an. Das Deaktivieren dieser Optionen stellt die vorher ausgewählte Farbe wieder her oder verwendet die Systemfarbe für das Quelltextelement, falls zuvor keine Farbe ausgewählt wurde.
Hintergrund	Zeigt das ausgewählte Quelltextelement in der Standardsystemfarbe für den Hintergrund an. Das Deaktivieren dieser Optionen stellt die vorher ausgewählte Farbe wieder her oder verwendet die Systemfarbe für das Quelltextelement, falls zuvor keine Farbe ausgewählt wurde.
Element	Legt die Syntaxhervorhebung für ein bestimmtes Quelltextelement fest. Sie können das gewünschte Element in der Liste Element auswählen oder im Beispiel-Quelltext-Editor darauf klicken. Beim Ändern der Farben der verschiedenen Sprachelemente werden die Auswirkungen im Beispiel-Quelltextfenster angezeigt.
Vordergrundfarbe	Legt die Vordergrundfarbe für das ausgewählte Code-Element fest. Die Vordergrundfarbe ändert sich automatisch, wenn Sie im Listenfeld Element ein anderes Element auswählen.
Hintergrundfarbe	Legt die Hintergrundfarbe für das ausgewählte Quelltextelement fest.
Anzeigebereich der jeweiligen Sprache	Klicken Sie auf das Register der jeweiligen Sprache, um anzuzeigen, wie sich Ihre Auswahl auf die Darstellung des Quelltextes auswirkt.

Anmerkung: Die Vorder- und Hintergrundfarbe des Elements Geänderte Zeile in der Liste Element sind die Farben, die dafür verwendet werden, die Zeilen zu markieren, die seit dem letzten Speichern geändert wurden, sowie Zeilen, die in der aktuellen Sitzung geändert und gespeichert wurden.

2.370 Pfade und Verzeichnisse (C++)

Tools > Optionen > Umgebungsoptionen > C++-Optionen > Pfade und Verzeichnisse

Verwenden Sie die Seite Pfade und Verzeichnisse, um für alle Packages Verzeichnis-, Compiler- und Linker-Optionen festzulegen.

Element	Beschreibung
Include-Pfad	Gibt die Verzeichnisse mit den Header-Dateien an, die standardmäßig für C++-Projekte verwendet werden.
Bibliothekspfad	Gibt an, wo die C++-Header- und Bibliotheksdateien für installierte Komponenten und Package zu finden sind.
Package-Ausgabeverzeichnis	Gibt an, wo der Compiler die compilierten Package-Dateien ablegt.
BPI/LIB-Ausgabeverzeichnis	Gibt an, wo die Package BPI - und LIB -Dateien standardmäßig platziert werden. LIB ist eine statische Bibliothek und BPI ist eine Importbibliothek. Diese Angaben können für einzelne Packages über die Package-Optionen geändert werden.
Suchpfad	Gibt die Verzeichnisse an, in denen der Projekt-Browser nach Unit-Dateien sucht, wenn im Suchpfad oder Quellpfad für das Projekt kein Bezeichner angegeben ist.
Refactoring-Pfad einschränken	Legt die Verzeichnisse fest, die nicht in das Refactoring einbezogen werden sollen. Wenn Sie beispielsweise ein Symbol umbenennen möchten, und eine Referenz darauf in einer Datei eines hier angegebenen Pfades gefunden wird, wird das Refactoring mit einer Fehlermeldung beendet. Zwei integrierte Verzeichnisse sind automatisch aus dem Refactoring ausgeschlossen: \$(BDS)\include\vcl und \$(BDS)\include\dinkumware . Dadurch wird verhindert, dass VCL- und Dinkumware STL-Quelldateien einem Refactoring unterzogen werden. Wenn Sie Bibliotheken von Fremdherstellern (wie z.B. Boost) verwenden, geben Sie in dieses Feld den Pfad zu den Bibliotheks-Header-Dateien ein.

Tip: Mit Hilfe der Umgebungsvariablen **\$(BCB)** können Sie einen Pfad relativ zum RAD Studio-Stamm-Verzeichnis angeben.

2.371 Typbibliothek (C++)

Tools > Optionen > Umgebungsoptionen > C++-Optionen > Typbibliothek

Verwenden Sie die Seite Typbibliothek dazu, die Optionen für den Editor Typbibliothek auszuwählen.

Element	Beschreibung
dispinterfaces in Element-Wrappern benutzen	Ist dieses Kontrollkästchen markiert, macht der Importer dispinterface zur Standardschnittstelle für die Komponente, sofern diese sowohl vtable- als auch IDispatch-basierte Schnittstellen unterstützt. Standardeinstellung ist die Benutzung der vtable-basierten Schnittstelle.
Präfixe für Eigenschafts-Getter/Setter im MS-Stil verwenden	Ist dieses Kontrollkästchen markiert, benutzt der Importer Präfixe für Eigenschafts-Getter/Setter-Methoden im Stil von Microsoft Visual C++. Andernfalls werden die Präfixe <code>get_</code> und <code>set_</code> benutzt.
Suffix ändern	Der Typbibliothek-Importer fügt das Suffix <code>.OCX</code> an die von ihm generierten Komponenten-Wrapper-Dateien an. Sie können dieses Verhalten ändern, indem Sie auf Suffix ändern klicken und ein neues Suffix in das Textfeld eingeben.
Spezielle CoClass-Flags beim Importieren ignorieren	Beim Importieren eines ActiveX-Steuerelements werden nur CoClasses in die Typbibliothek übernommen, die nicht als Hidden, Restricted oder Predefined, sondern als Can Create markiert sind. Diese Flags sollten gesetzt werden, wenn das Objekt für allgemeine Zwecke verwendet wird. Wenn Sie allerdings ein Steuerelement nur für eine interne Anwendung erstellen möchten, können Sie die Flags überschreiben, um CoClass-Wrappers zu generieren. In diesem Fall würden Sie für Spezielle CoClass-Flags beim Importieren ignorieren Hidden und Restricted markieren sowie Can Create deaktivieren. Markieren Sie die Flags, die beim Import von ActiveX-Steuerelementen ignoriert werden sollen.
Predefined	Client-Anwendungen sollten automatisch eine einzelne Instanz dieses Objekts erzeugen.
Restricted	Eine als restricted markierte CoClass wird von Tools ignoriert, die auf COM-Objekte zugreifen. Sie wird von der Typbibliothek zur Verfügung gestellt, der Zugriff muss jedoch autorisiert sein.
Verborgen	Das Interface existiert, sollte aber in einem benutzerorientierten Browser nicht angezeigt werden.
Can Create	Das Interface kann mit <code>CoCreateInstance</code> erzeugt werden.

2.372 Debugger-Optionen

Tools > Optionen > Debugger-Optionen

Verwenden Sie diese Seite zum Festlegen von allgemeinen Debugger-Optionen für die IDE.

Element	Beschreibung
Integriertes Debuggen	Aktiviert den integrierten Debugger.
TD32-Tasten	Ermöglicht die Verwendung von TD32-Tasten (32-Bit-Turbo-Debugger) in der IDE. Wenn diese Option markiert ist, ist die TD32-Tastenzuordnung während einer Debug-Sitzung aktiv. Beachten Sie, dass wenn diese Option ausgewählt wird, auch automatisch die Option Puffer schreibgeschützt aktiv ist und sich nicht deaktivieren lässt.
Puffer schreibgeschützt	Markiert alle Editordateien, einschließlich Projekt- und Arbeitsgruppdateien, bei der Programmausführung als schreibgeschützt. Wenn diese Option ausgewählt ist, werden die Dateiattribute nach der Programmausführung nicht geändert. Andernfalls erhalten die Dateiattribute nach der Programmausführung wieder die ursprüngliche Konfiguration.
Beim Start Debugger-Menü oben	Verlagert den Debugger-Bereich im Kontextmenü des Quelltext-Editors zum einfacheren Zugriff auf die Debugger-Befehle nach oben, wenn ein Programm in der IDE ausgeführt wird. Zeigt das Kontextmenü des Quelltext-Editors nach einem Rechtsklick auf eine beliebige Stelle im Quelltext-Editor an.
Beim Debuggen implizit geöffnete Dateien automatisch schließen	Schließt alle Dateien, die nicht explizit geöffnet wurden. Wenn der Debugger eine Datei implizit geöffnet hat, wird diese Datei am Ende der Debug-Sitzung automatisch geschlossen, sofern sie nicht geändert wurde. Dateien, die bearbeitet wurden oder für die ein Haltepunkt gesetzt wurde, werden nicht automatisch geschlossen.

2.373 VCL-Designer

Tools ▶ Optionen ▶ VCL-Designer

Verwenden Sie diese Seite zum Festlegen von Präferenzen für den Designer für VCL-Formulare.

Element	Beschreibung
Raster anzeigen	Zeigt im Designer ein Raster mit Punkten an, um die Ausrichtung von Steuerelementen zu erleichtern.
Designer-Richtlinien verwenden	Aktiviert Richtlinien im Formular-Designer. Richtlinien vereinfachen die Ausrichtung von Komponenten in einem Formular.
Am Raster ausrichten	Richtet Steuerelemente im Designer automatisch an der nächsten Rasterlinie aus, wenn Sie ein Steuerelement verlagern.
Rastergröße/Ausrichtungstoleranz	Setzt den Pixel-Abstand zwischen den Rasterlinien entlang der X- und Y-Achse. Legen Sie eine Zahl (zwischen 2 und 128) fest, um den Abstand zu erhöhen.
Komponenten-Titel zeigen	Zeigt die Titel von nicht-visuellen Komponenten an, die Sie einem Formular oder Datenmodul hinzufügen.
Designer-Hinweise anzeigen	Zeigt einen Klassennamen in den Kurzhinweisen für nicht-visuelle Komponenten in einem Formular oder Datenmodul an.
Erweiterte Hinweise anzeigen	Zeigt Kurzhinweise für Steuerelemente mit Ursprung (Position im Formular), Größe (Breite und Höhe), Tabstop (ob dem Steuerelement mit der Tabulatortaste der Fokus zugewiesen werden kann) und der Reihenfolge, in der das Steuerelement dem Formular hinzugefügt wurde, an. Wenn Designer-Hinweise anzeigen nicht markiert ist, ist diese Option deaktiviert.
Eingebetteter Designer	Zeigt VCL-Formulare auf der Registerseite Design neben der Registerseite Code an. Wenn diese Option nicht markiert ist, werden VCL-Formulare als frei-platzierbare, nicht-angedockte Fenster angezeigt. Das Formular und der Quelltext können so gleichzeitig eingesehen werden.
Virtuelle Bildschirmposition anzeigen	Zeigt die Ansicht virtuelle Bildschirmposition in der unteren rechten Ecke des Formular-Designers an. Sie können diese Ansicht dazu verwenden, die Laufzeitposition des Formulars am Bildschirm schnell zu setzen.
Neue Formulare als Text	Legt das Format fest, in dem Formulardateien gespeichert werden. Die Formulardateien eines Projekts können im binären oder im Textformat gespeichert werden. Textdateien lassen sich in anderen Programmen bearbeiten und von einem Versionskontrollsystem verwalten. Binärdateien sind hingegen mit früheren Versionen des Produkts kompatibel. Sie können diese Einstellung bei einzelnen Formularen außer Kraft setzen, indem Sie mit der rechten Maustaste klicken und die Option Text-DFM oder Text-XFM aktivieren oder deaktivieren.
Autom. Formulare & Datenmodule	Legt fest, ob alle Formulare beim Programmstart automatisch erstellt werden. Ist die Option deaktiviert, werden die nach dem ersten Formular (dem Hauptformular) zu einem Projekt hinzugefügten Formulare nicht in die Liste Automatisch erzeugen, sondern in die Liste Verfügbare Formulare aufgenommen. Wo ein Formular aufgelistet wird, lässt sich ändern, indem Sie Projekt ▶ Optionen ▶ Formulare wählen.

2.374 Bibliothek

Tools ▶ Optionen ▶ Delphi-Optionen ▶ Bibliothek

Verwenden Sie diese Seite, um für alle Packages Verzeichnis-, Compiler- und Linker-Optionen festzulegen.

Element	Beschreibung
Bibliothekspfad	Gibt die Suchpfade für die Quelltextdateien des Package an. Der Compiler berücksichtigt nur die Dateien in diesen Verzeichnissen. Wenn Sie eine Datei in einem anderen Verzeichnis verwenden, wird ein entsprechender Compiler-Fehler ausgegeben.
Package-Ausgabeverzeichnis	Gibt an, wo der Compiler die compilierten Package-Dateien ablegt.
DCP/DCPIL-Ausgabeverzeichnis	Legt ein separates Verzeichnis für die <code>.dcp</code> - (Win32) oder <code>.dcplib</code> - (.NET) Dateien fest.
Suchpfad	<p>Gibt die Verzeichnisse an, in denen der Projekt-Browser nach Unit-Dateien sucht, wenn im Suchpfad oder Quellpfad für das Projekt kein Bezeichner angegeben ist.</p> <p>Bei Win32- und .NET-Delphi-Projekten werden die mit dieser Option angegebenen Verzeichnisse an den Debug-Suchpfad für das Projekt angehängt. Die Debug-Suchreihenfolge für Unit-Dateien wird daher durch die folgenden Pfadeinstellungen bestimmt:</p> <ol style="list-style-type: none"> 1. Projektspezifischer Pfad für Debugger, der auf der Seite Projekt ▶ Optionen ▶ Debugger angegeben wurde. 2. Suchpfad (diese Option). 3. Der globale Pfad für Debugger, der auf der Seite Tools ▶ Optionen ▶ Debugger-Optionen ▶ Borland-Debugger angegeben wurde. <p>Bei C#- und Visual Basic-Projekten oder wenn kein Projekt in der IDE geladen ist, wird nur der auf der Seite Tools ▶ Optionen ▶ Debugger-Optionen ▶ Borland-Debugger unter Pfad für Debugger angegebene Pfad nach Quelldateien durchsucht.</p>
Namespace-Präfixe	Legt die Präfixe für Namespaces in der Punktnotation fest, um das Erstellen von Kurzversionen von Namespaces in den <code>uses</code> -Klauseln des Quelltextes zu ermöglichen. Z.B. können Sie anstatt <code>Borland.Vcl.DB</code> zu schreiben, das Namespace-Präfix <code>Borland.Vcl</code> festlegen. In der <code>uses</code> -Klausel können Sie dann <code>uses DB</code> ; verwenden.
Debug-DCU/DCUIL-Pfad	Um diese Option verwenden zu können, müssen Sie die Option Mit Debug-DCU/DCUILs auf der Seite Projekt ▶ Optionen ▶ Compiler setzen. Der Debugger sucht dann erst in diesem Pfad nach <code>.dcu</code> - (Win32) oder <code>.dcuil</code> - (.NET)-Dateien, bevor der Unit-Suchpfad berücksichtigt wird.

Tip: Mehrere Werte im Eingabefeld trennen Sie mit einem Semikolon voneinander. Um mehrere Werte mithilfe eines Dialogfeldes hinzuzufügen, können Sie auch alternativ die Ellipsen-Schaltfläche neben dem Eingabefeld anklicken. Zum Festlegen von Betriebssystemumgebungsvariablen in einem Eingabefeld verwenden Sie die Syntax `$(VariablenName)`.

2.375 Anzeigeoptionen

Tools > Optionen > Editor-Optionen > Anzeige

Verwenden Sie diese Seite zum Setzen von Anzeige- und Schriftoptionen für den Quelltext-Editor.

Element	Beschreibung
BRIEF Cursorform	Verwendet die Cursorformen des BRIEF-Editors.
Volle Bildschirmgröße	Maximiert den Quelltext-Editor auf die gesamte Bildschirmgröße. Wenn diese Option deaktiviert ist, verdeckt der Quelltext-Editor im maximierten Zustand das Hauptfenster nicht.
Popup-Seitenmenü sortieren	Sortiert die Listen der angezeigten Seiten alphabetisch, wenn Sie auf ein Register im Quelltext-Editor rechtsklicken und Seiten auswählen. Wenn diese Option deaktiviert ist, werden die Seiten in der Reihenfolge ihrer Erstellung sortiert.
Bilder auf Register anzeigen	Zeigt auf jedem Register im Quelltext-Editor ein Symbol an.
Sichtbarer rechter Rand	Zeigt am rechten Rand des Quelltext-Editors eine vertikale Linie an.
Zeilennummern anzeigen	Zeigt die aktuelle Zeilennummer und jede zehnte Zeilennummer am linken Rand des Quelltext-Editors an.
Alle Zeilen numerieren	Zeigt am linken Rand des Quelltext-Editors Zeilennummern an.
Leiste sichtbar	Zeigt am linken Rand des Quelltext-Editors eine Leiste an.
Rechter Rand	Setzt den rechten Rand des Quelltext-Editors. Die Vorgabe beträgt 80 Zeichen.
Breite der Leiste	Setzt die Breite der Leiste. Die Vorgabe ist 30.
Editorschrift	Wählen Sie aus den auf Ihrem System installierten Bildschirmschriften (die in der Liste angezeigt werden) eine Schriftart aus. Der Quelltext-Editor verwendet und zeigt nur nicht-proportionale Bildschirmschriftarten, wie z.B. Courier, an. Beispieltext wird im Feld Beispiel angezeigt.
Schriftgrad	Wählen Sie für die im Listenfeld Editorschrift ausgewählten Schrift eine Schriftgröße aus den vordefinierten Schriftgrößen aus. Beispieltext wird im Feld Beispiel darunter angezeigt.
Beispiel	Zeigt ein Beispiel der ausgewählten Schriftart und -größe an.

2.376 ECO - Allgemeine Optionen

Tools > Optionen > ECO - Allgemeine Optionen

Verwenden Sie diese Seite, um die ECO-Framework-Einstellungen festzulegen.

Element	Beschreibung
Automatische Compilierung	Compiliert die Anwendung jedes Mal, wenn die ECO-Entwurfszeitumgebung eine compilierende Anwendung benötigt, um ein aktuelles Modell wiederzugeben. Diese Option wird bei den Operationen Datenbank erzeugen, Datenbankentwicklung, Modell validieren und Packages hinzufügen angewendet. Die Compilierung wird nur bei Bedarf durchgeführt.
EcoSpace-Komponenten automatisch anbinden	<p>Setzt die folgenden Eigenschaften automatisch:</p> <p>Wenn Sie eine PersistenceMapper-Komponente (für BDP, SQL oder XML) in einem ECOSpace-Designer platzieren, wird die Eigenschaft PersistenceMapper des ECO-Space automatisch gesetzt.</p> <p>Wenn Sie eine Datenverbindungs-Komponente in einem ECO-Space platzieren und die Eigenschaft PersistenceMapper gesetzt ist, wird die Eigenschaft Connection des PersistenceMapper gesetzt, wenn sie denselben Typ (PersistenceMapperSqlServer und SqlConnection oder PersistenceMapperBdp und BdpConnection) haben.</p> <p>Diese Eigenschaften werden nur beim Einfügen automatisch gesetzt und nur wenn die betreffende Eigenschaft leer ist, bevor die neue Komponente eingefügt wird.</p>
Task-Hinweise im ECO-Space-Designer anzeigen	Zeigt eine Task-Liste an, wenn Sie die Maus über Komponenten im ECO-Space-Designer bewegen. Neben jedem beendeten Task wird ein 'X' angezeigt.
ECO-Quelltext synchronisieren	Ist diese Option aktiv, wird der Quellcode automatisch erneuert, während Sie mit den ECO-Klassendiagramm und dem ECO-Zustandsdiagramm arbeiten. Ist diese Option nicht aktiv, können Sie zuerst Änderungen am Modell vornehmen und dann den gesamten Quelltext aktualisieren, indem Sie das Kontextmenü der Modellansicht öffnen.
Debug-Code erzeugen	Ist diese Option aktiv, wird eine Assertion hinzugefügt, um die Getter- und Setter-Methoden zuzuordnen. Die Assertion gibt eine informativere Fehlermeldung aus, wenn versucht wird, auf ein Attribut zuzugreifen, dessen Wert Null ist.
Standard-Pluralsuffix	<p>Diese Option wird bei nicht benannten Assoziationsenden verwendet, wenn die Multizipitität des Assoziationsendes zu groß ist (*) .</p> <p>ECO konstruiert den Namen des Assoziationsendes, indem es das Pluralsuffix an den Namen der Klasse an das Ende der zu hohen Assoziation anhängt.</p> <p>Diese Option kann in jedem ECO-Package überschrieben werden, das Sie erstellt haben, indem Sie die Eigenschaft Plural-Suffix des Package im Objektinspektor gesetzt haben.</p>
C#-Quelltexterzeugung: Eine Datei pro Klasse	Wenn diese Option aktiv ist, erstellt die IDE getrennte Quelldateien für jede ECO-Klasse im Modell. Andernfalls wird der Code für die ECO-Klasse in der Quelldatei des ECO-Package erzeugt, zu der die Klasse gehört.
Delphi-Quelltexterzeugung: Eine Datei pro Klasse	Wenn diese Option aktiv ist, erstellt die IDE getrennte Quelldateien für jede ECO-Klasse im Modell. Andernfalls wird der Code für die ECO-Klasse in der Quelldatei des ECO-Package erzeugt, zu der die Klasse gehört.
Muster für Unit-Name	Das Muster für den Unit-Namen legt den Namen der Delphi-Unit für neue ECO-Klassen fest. Der Standard-Mustername ist {0}Unit, wobei {0} der Klassename ist.

Siehe auch

[Überblick zum ECO-Framework](#)

Überblick zu ECO-Modellierungstools

Anwendungen mit dem ECO-Framework erstellen

ECO-Quelltext erneut generieren und aktualisieren

2.377 Editor-Optionen

Tools > Optionen > Editor-Optionen

Verwenden Sie diese Seite, um das Verhalten des Quelltext-Editors anzupassen.

Element	Beschreibung	
Einfügemodus	Fügt Text am Cursor ein, ohne vorhandenen Text zu überschreiben. Wenn die Option Einfügemodus deaktiviert ist, wird der Text am Cursor überschrieben. (Verwenden Sie die Taste Einfg , um den Einfügemodus im Quelltext-Editor zu aktivieren/deaktivieren, ohne diese Standardeinstellung zu ändern.)	
Gruppe rückgängig	Macht den letzten Editierbefehl sowie alle nachfolgenden Editierbefehle desselben Typs rückgängig, wenn Sie Alt+Rück drücken oder Bearbeiten > Rückgängig auswählen.	
Cursor hinter EOF	Positioniert den Cursor an das Dateiende.	
Doppelklick markiert Zeile	Markiert die Zeile, wenn Sie auf ein Zeichen in der Zeile doppelklicken. Falls deaktiviert, wird nur das ausgewählte Wort markiert.	
Ausschneiden/Einfügen immer aktiv	Aktiviert die Befehle Bearbeiten > Ausschneiden und Bearbeiten > Kopieren auch dann, wenn kein Text markiert ist	
Suchtext auto-vervollständigen	Aktiviert die Vervollständigungsfunktion im Suchen-Dialog.	
Sicherungsdateien erstellen	Erstellt jedes Mal eine Sicherungsdatei, wenn Sie eine Datei in der IDE aktualisieren und speichern. Sicherungsdateien werden in einem verborgenen Unterverzeichnis namens __history des im aktuellen Verzeichnisses gespeichert und können auf der Registerseite Versionsgeschichte verwaltet werden. Im Feld Anzahl Dateisicherungen können Sie die Anzahl der Sicherungsdateien festlegen, die im Verzeichnis __history gespeichert werden soll.	
Rückgängig Speichern	nach	Ermöglicht die Rücknahme von Änderungen nach dem Speichern.
BRIEF reguläre Ausdrücke		Verwendet BRIEF reguläre Ausdrücke. Reguläre Ausdrücke sind bei Operationen mit Mustervergleichen hilfreich.
Persistente Blöcke		Behält die Markierung eines Blocks auch dann bei, wenn der Cursor bewegt wird, bis Sie einen neuen Block auswählen.
Blöcke überschreiben		Ersetzt einen markierten Textblock durch das, was Sie als nächstes eingeben. Falls außerdem die Option Persistente Blöcke markiert ist, wird der eingegebene Text an das Ende des derzeit markierten Blocks angefügt.
Text am Cursor suchen		Platziert den Text am Cursor in das Listenfeld Suchen nach des Dialogfelds Text suchen, wenn Sie Suchen > Suchen auswählen. Wenn diese Option deaktiviert ist, müssen Sie den zu suchenden Text manuell eingeben, sofern das Listenfeld Suchen nach leer ist. Enthält das Listenfeld bereits einen Text, fügt ihn der Editor am Cursor ein.
Zeilenenden erhalten		Erhält das Zeilenende.

Editor-Einstellung	<p>Stellt eine schnelle Möglichkeit bereit, den Editor anhand von vorkonfigurierten Standardeinstellungen aus der Dropdown-Liste zu konfigurieren.</p> <p>Standard verwendet Tastenzuordnungen, die den CUA-Zuordnungen entsprechen (Vorgabe).</p> <p>IDE - Klassisch verwendet Tastenzuordnungen, die den Borland Classic-Editor-Tastatureingaben entsprechen.</p> <p>BRIEF-Emulation verwendet Tastenzuordnungen, die die meisten BRIEF-Standardtastatureingaben emulieren.</p> <p>Epsilon-Emulation verwendet Tastenzuordnungen, die einen Großteil des Epsilon-Editors emulieren.</p> <p>Visual Studio-Emulation verwendet Tastenzuordnungen, die einen Großteil des Visual Studio-Editors emulieren.</p> <p>Visual Basic-Emulation verwendet Tastenzuordnungen, die einen Großteil des Visual Basic-Editors emulieren.</p>
Anzahl Rückgängig	<p>Legt die Anzahl der Tastenanschläge fest, die rückgängig gemacht werden können. Der Standardwert ist 32.767.</p> <p>Der Rückgängig-Puffer wird jedes Mal bei der Codegenerierung gelöscht.</p>
Anzahl Dateisicherungen	<p>Wenn Sicherungsdateien erstellen markiert ist, können Sie hier festlegen, wie viele Sicherungsdateien für in der IDE aktualisierte und gespeicherte Dateien aufbewahrt werden sollen. Der Vorgabewert ist 10, Sie können aber einen Wert von 1 bis 90 angeben.</p> <p>Durch das Verringern der Anzahl Dateisicherungen werden vorhandene Sicherungsdateien nicht aus dem Verzeichnis _history gelöscht.</p>

Siehe auch

Standard-Tastaturvorlage ([siehe Seite 1335](#))

Tastaturvorlage IDE - Klassisch ([siehe Seite 1338](#))

Tastaturvorlage BRIEF-Emulation ([siehe Seite 1340](#))

Tastaturvorlage Epsilon-Emulation ([siehe Seite 1342](#))

Tastaturvorlage Visual Studio-Emulation ([siehe Seite 1344](#))

Tastaturvorlage Visual Basic-Emulation ([siehe Seite 1345](#))

2.378 Umgebungsoptionen

Tools > Optionen > Umgebungsoptionen

Verwenden Sie diese Seite, um die IDE-Konfigurationsvorgaben festzulegen.

Element	Beschreibung
Editordateien	Speichert alle im Quelltext-Editor befindlichen, geänderten Dateien, wenn Sie das Projekt ausführen, compilieren, erstellen, oder wenn Sie das Programm beenden.
Projekt-Desktop	Speichert die Anordnung auf Ihrem Desktop, wenn Sie ein Projekt schließen oder die IDE schließen. Wenn Sie dasselbe Projekt später erneut öffnen, werden alle Dateien, die beim letzten Schließen geöffnet waren, wieder geöffnet, auch wenn sie vom Projekt nicht verwendet werden.
Compiler-Fortschritt anzeigen	Zeigt beim Compilieren des Programms Informationen über den Status der Operation an.
Beim Start als Symbol	Verkleinert die IDE zum Symbol, wenn Sie die Anwendung mit Start > Start ausführen. Wenn Sie die Anwendung schließen, wird die IDE in der ursprünglichen Größe wiederhergestellt. Wenn Sie eine Anwendung ohne den Debugger ausführen, bleibt die IDE minimiert.
Designer zur Laufzeit verbergen	Verbirgt alle Designer-Fenster (z.B. den Objektinspektor und die Ausrichtungspalette), während die Anwendung ausgeführt wird. Die Fenster werden wieder angezeigt, wenn Sie die Anwendung schließen.
Befehlszeile anzeigen	Zeigt beim Compilieren eines Projekts den für die Compilierung verwendeten Befehl im Fenster Meldungen an. In einer C#-Umgebung wird der für die Compilierung verwendete Befehl und der Inhalt der Antwortdatei angezeigt. Die Antwortdatei listet die Compiler-Optionen und die zu compilierenden Quelldateien auf.
Verbosity	Legt die Verbosity-Ebene der Build-Ausgabe fest. Wählen Sie Quiet, Minimal, Normal, Detailliert oder Diagnose. Die Build-Ausgabe wird auf der Registerkarte Ausgabe des Fensters Meldungen angezeigt.
Autom. Andocken beim Ziehen	Ermöglicht, Tool-Fenster anzudocken, indem Sie den Umriss eines Fensters über ein anderes Fenster ziehen. Wenn diese Option eingeschaltet ist, können Sie sie während des Ziehens mit der Taste STRG deaktivieren. Ist diese Option ausgeschaltet, wird sie mit der Taste STRG aktiviert.
Gemeinsame Objektablage	Legt den Verzeichnispfad fest, unter dem das Projekt nach der gemeinsamen Objektablage sucht. Klicken Sie auf die Schaltfläche Durchsuchen, um nach Verzeichnissen zu suchen.
Standardprojekt	Legt den Verzeichnispfad fest, unter dem das Projekt nach den Standardprojektdateien sucht. Klicken Sie auf die Schaltfläche Durchsuchen, um nach Verzeichnissen zu suchen.

2.379 Umgebungsvariablen

Tools▶**Optionen**▶**Umgebungsoptionen**▶**Umgebungsvariablen**

Verwenden Sie diese Seite, um Systemumgebungsvariablen anzuzeigen und um vom Anwender überschriebene Variablen zu erstellen, zu bearbeiten und zu löschen.

Element	Beschreibung
Systemvariablen	Zeigt eine Liste aller Umgebungsvariablen und ihrer Werte an, die auf Systemebene definiert sind. Eine Systemvariable kann nicht gelöscht, aber überschrieben werden.
Variable überschreiben	Öffnet das Dialogfeld Variable überschreiben, in dem Systemvariablen geändert werden können, um neue vom Anwender überschriebene Variablen zu erstellen. Diese Schaltfläche ist deaktiviert (abgedunkelt), bis Sie eine Variable in der Liste Systemvariablen auswählen.
Vom Anwender überschrieben	Zeigt alle Anwendervariablen mit den zugehörigen Werten an. Anwendervariablen haben Vorrang vor Systemvariablen und verlieren ihre Gültigkeit erst, wenn sie gelöscht werden.
Neu	Öffnet das Dialogfeld Neue Anwendervariable, in dem Sie eine neue Anwendervariable aus einer Systemvariable erstellen können.
Bearbeiten	Öffnet das Dialogfeld Anwendervariable bearbeiten, in dem Sie die Anwendervariable ändern können, die in der Liste Vom Anwender überschrieben ausgewählt ist.
Löschen	Löscht die Anwendervariable, die in der Liste Vom Anwender überschrieben ausgewählt ist.

2.380 Ereignisprotokoll-Optionen

Tools > Optionen > Debugger-Optionen > Ereignisprotokoll

Verwenden Sie dieses Dialogfeld zum Festlegen des Inhalts, der Größe und des Erscheinungsbildes des Ereignisprotokolls.

Element	Beschreibung
Beim Start leeren	Löscht das Ereignisprotokoll zu Beginn jeder Debugging-Sitzung. Wenn diese Option beim Testen mehrerer Prozesse aktiviert ist, wird das Ereignisprotokoll beim Start des ersten Prozesses gelöscht. Wenn dann weitere Prozesse gestartet werden und mindestens einer bereits getestet wird, erfolgt keine Löschung des Ereignisprotokolls.
Länge unbegrenzt	Entfernt die Längenbegrenzung für das Ereignisprotokoll. Wenn diese Option nicht markiert ist, müssen Sie im Feld Länge die maximale Länge des Ereignisprotokolls angeben.
Länge	Gibt die maximale Länge des Ereignisprotokolls an. Wenn Sie Länge unbegrenzt aktiviert haben, kann diese Option nicht gewählt werden. Beim Testen mehrerer Prozesse gilt der hier angegebene Wert für das gesamte Ereignisprotokoll, nicht für einen Einzelprozess. Die Standardeinstellung ist 100.
Prozessinfo mit Ereignis anzeigen	Zeigt bei jedem Ereignis Name und ID des Prozesses an, durch den es generiert wurde.
Neue Ereignisse in Ansicht	Steuert die Anzeige im Ereignisprotokoll. Deaktivieren Sie diese Option, wenn die Ansicht nicht beim Auftreten neuer Ereignisse verschoben werden soll. (Diese Option ist standardmäßig aktiviert.)
Haltepunktmeldungen	Schreibt jedes Mal, wenn bei der Programmausführung ein Haltepunkt oder eine erste Zufalls-Exception erreicht wird, eine Meldung in das Ereignisprotokoll. Die Meldung enthält die aktuelle EIP-Adresse des getesteten Programms und Informationen zum Haltepunkt (Durchlaufzähler, Bedingung, Name der Quelldatei und Zeilennummer) bzw. zur Exception. (Diese Option ist standardmäßig aktiviert.)
Prozessmeldungen	Schreibt eine Meldung in das Ereignisprotokoll, wenn ein Prozess geladen bzw. beendet wird und wenn ein Modul vom Prozess geladen bzw. aus dem Speicher entfernt wird. (Diese Option ist standardmäßig aktiviert.)
Thread-Meldungen	Schreibt eine Meldung in das Ereignisprotokoll, wenn ein Thread in einer Debug-Sitzung erstellt oder freigegeben wird. (Diese Option ist standardmäßig aktiviert.)
Modulmeldungen	Schreibt eine Meldung in das Ereignisprotokoll, wenn ein Modul (ausführbar, gemeinsam genutztes Objekt oder Package) geladen oder beendet wird. Die Meldung enthält den Namen des Moduls, seine Basisadresse und ob Debug-Informationen vorhanden sind. (Diese Option ist standardmäßig aktiviert.)
Ausgabemeldungen	Schreibt eine Meldung in das Ereignisprotokoll, wenn das Programm oder eines der zugehörigen Module OutputDebugString aufruft. (Diese Option ist standardmäßig aktiviert.) Diese Einstellung wird nur vom CodeGear Win32-Debugger verwendet.
Meldungen der Anwendungsdomäne	Schreibt immer dann eine Meldung in das Ereignisprotokoll, wenn eine Anwendungsdomäne erstellt oder geschlossen wird. Die Meldung zum Erstellen der Anwendungsdomäne erscheint vor den Meldungen für die Anwendung. Die Meldung zum Erstellen der Anwendungsdomäne erscheint vor den Prozess-Lade-Meldungen für die Anwendung. (Diese Option ist standardmäßig aktiviert.) Diese Einstellung wird nur vom CodeGear .NET-Debugger verwendet.
Verwaltete Debug-Assistentmeldungen	Schreibt immer dann eine Meldung in das Ereignisprotokoll, wenn ein verwalteter Debug-Assistent ausgelöst wird. (Diese Option ist standardmäßig aktiviert.) Diese Einstellung wird nur vom CodeGear .NET-Debugger verwendet. Weitere Informationen über verwaltete Debug-Assistenten finden Sie bei MSDN unter http://msdn2.microsoft.com/en-us/library/d21c150d.aspx .

Fenstermeldungen		Schreibt für jede Fenstermeldung, das zu einem der Anwendungsfenster gesendet oder dort angezeigt wird, eine Meldung in das Ereignisprotokoll. Im Ereignisprotokoll sind Einzelheiten zur Meldung (Meldungsnname und alle relevanten Daten, die in den zugehörigen Parametern codiert sind) enthalten. Die Meldungen werden nicht unmittelbar in das Protokoll geschrieben, solange der Prozess läuft und im Debugger nicht angehalten wird. Sobald dies jedoch der Fall ist (an einem Haltepunkt oder wenn Sie Start▶Pause wählen), werden die Meldungen in das Ereignisprotokoll geschrieben. (Ist per Voreinstellung deaktiviert.) Diese Einstellung wird nur vom CodeGear Win32-Debugger verwendet.
Farben Ereignisprotokolls verwenden	des	Ermöglicht, verschiedene Typen von Ereignismeldungen im Ereignisprotokoll farbig anzuzeigen.
Vordergrund		Legt die Farbe für den Text im Ereignisprotokoll fest.
Hintergrund		Legt die Farbe für den Hintergrund im Ereignisprotokoll fest.

2.381 Explorer

Tools > Optionen > Umgebungsoptionen > Explorer

Verwenden Sie diese Seite, um das Verhalten des Struktur-Fensters und der Projektverwaltung zu steuern.

Anmerkung: Klicken Sie ein Element im Struktur-Fenster mit der rechten Maustaste an, und wählen Sie Eigenschaften, um diese Seite als separates Dialogfeld Explorer-Optionen zu öffnen.

Element	Beschreibung
Unvollständige Klassen hervorheben	Zeigt unvollständige Eigenschaften und Methoden im Struktur-Fenster fett an. (Nicht für die C++-Entwicklung verwendbar.)
Deklarationssyntax anzeigen	Zeigt die Syntax und den Typ der Methoden oder Eigenschaften an. Standardmäßig werden nur die Namen der Quelltextelemente im Struktur-Fenster angezeigt. (Nicht für die C++-Entwicklung verwendbar.)
Explorer-Sortierung: Alphabetisch	Führt Quelltextelemente im Struktur-Fenster alphabetisch auf.
Explorer-Sortierung: Quelle	Zeigt die Quelltextelemente in der Reihenfolge an, in der sie in der Quelltextdatei deklariert sind.
Vervollständigung von Klassen: Unvollständige Eigenschaften vervollständigen	Bei einer Eigenschaftsdeklaration wird der Rest der Definition zum Lesen und Schreiben der Eigenschaft vervollständigt. Ist die Option deaktiviert, werden nur Methoden vervollständigt. (Nicht für die C++-Entwicklung verwendbar.)
Explorer-Kategorien	Steuert, in welchen Kategorien Quelltextelemente im Struktur-Fenster oder in der Projektverwaltung angezeigt werden. Wenn Sie eine Kategorie aktivieren, werden die Elemente dieses Typs unter einem einzelnen Knoten im Baumdiagramm angezeigt. Wenn Sie eine Kategorie deaktivieren, werden ihre Elemente unabhängig voneinander im Stamm der Baumhierarchie angezeigt. Die fett angezeigten Ordner werden zuerst berücksichtigt, wenn ein Element aufgrund eines Konflikts in zwei Ordner aufgenommen werden kann. So wird beispielsweise ein private Feld im private Ordner angezeigt, wenn die Kategorien Private und Feld aktiviert sind. Bei aktiviertem Ordner ist an einem Symbol rechts neben dem Kontrollkästchen zu erkennen, ob der Ordner erweitert ist. Durch Klicken auf dieses Symbol können die Ordner im Struktur-Fenster geöffnet und geschlossen werden. Die vorgenommenen Änderungen werden wirksam, sobald Sie auf OK klicken.

2.382 Generische sortierte Liste

Verwenden Sie dieses generische Dialogfeld zum Verwalten von Listen mit Einträgen, wie z.B. Pfade oder Definitionen.

Anmerkung: Nicht alle im Folgenden beschriebenen Optionen stehen in allen Vorkommen dieses Dialogfelds zur Verfügung.

Element	Beschreibung	
Sortierte Liste der ...	Führt die Einträge auf, die in der angezeigten Reihenfolge durchsucht werden.	
Pfeilsymbole	Verschiebt den ausgewählten Eintrag in der Liste nach oben.	
Textfeld	Gibt einen Eintrag an, der der Liste hinzugefügt oder ersetzt werden soll. Sie markieren einen Eintrag in der Liste, indem Sie ihn anklicken.	
Ellipse (...)	Zeigt ein Dialogfeld an, in dem Sie zu einem Ordner navigieren und ihn auswählen können. Der Eintag, den Sie auswählen, wird im Textfeld angezeigt.	
Ersetzen	Ersetzt den markierten Eintrag durch den Eintrag im Textfeld.	
Hinzufügen	Fügt das Element aus dem Textfeld in die Liste ein.	
Löschen	Entfernt den markierten Eintrag aus der Liste.	
Ungültige löschen	Pfade	Entfernt alle abgedunkelten Pfade aus der Pfadliste. Ein Pfad wird abgedunkelt dargestellt, wenn er nicht mehr gültig ist.

2.383 HTML/ASP.NET-Optionen

Tools > Optionen > HTML/ASP.NET-Optionen

Verwenden Sie dieses Dialogfeld zum Festlegen von Vorgaben für die Bearbeitung von HTML im Designer.

Element	Beschreibung
Raster anzeigen	Zeigt im Designer ein Raster mit Punkten an, um die Ausrichtung von Steuerelementen zu erleichtern.
Am Raster ausrichten	Richtet Steuerelemente im Designer automatisch an der nächsten Rasterlinie aus, wenn Sie ein Steuerelement verlagern.
Rastergröße	Setzt den Pixel-Abstand zwischen den Rasterlinien entlang der X- und Y-Achse. Legen Sie eine Zahl (zwischen 2 und 128) fest, um den Abstand zu erhöhen.
HTML-Steuerelemente mit Windows-Theme übergeben	Übernimmt auf der Design-Seite für HTML-Steuerelemente das aktuelle Windows-Theme (wenn Windows XP installiert und ein Theme aktiviert ist). Damit können Sie die Auswirkung eines Themes auf Steuerelemente während des Entwurfs feststellen. Während der Ausführung bestimmen die Theme-Einstellungen des Anwenders die Darstellung der Steuerelemente.
DIV-Tag einfügen...	Fügt ein <DIV>-Tag zur Kennzeichnung eines Abschnitts in die HTML-Datei ein, wenn Sie die Taste EINGABE drücken. Ansonsten wird ein <P>-Tag zur Kennzeichnung eines Absatzes eingefügt.
Standardseiten-Layout	Legt die Positionierung von hinzugefügten Komponenten fest. Fluss-Layout: Wenn eine Komponente abgelegt wird, wird sie ähnlich wie in Textverarbeitungssystemen von oben nach unten und von links nach rechts positioniert. Raster-Layout: Komponenten werden anhand der absoluten X- und Y-Koordinaten positioniert.
Smart Tasks beim Erstellen des Steuerelements automatisch anzeigen	Wenn diese Option aktiviert ist, wird nach dem Ablegen eines Steuerelements auf einem Web-Formular ein Smart-Task-Fenster angezeigt.
Designer-Element beim Bearbeiten im Tag-Editor markieren	Markiert das HTML-Steuerelement im Designer, wenn das zugehörige Tag im Tag-Editor bearbeitet wird.
Markierungsfarbe wählen	Wird aktiviert, wenn die Option Designer-Element beim Bearbeiten im Tag-Editor markieren ausgewählt ist. Öffnet ein Fenster zur Auswahl der Markierungsfarbe für HTML-Steuerelemente. Vorgabe ist gelb.
Web-Formulare	Wird nur für ASP.NET verwendet. Gibt an, was beim Erstellen eines Web-Dokuments im Bearbeitungsfester angezeigt wird. Designer: Zeigt eine WYSIWIG-Designer an. Markup-Editor: Zeigt einen Quelltext-Editor für Auszeichnungen, wie z.B. HTML- oder ASP-Quelltext, an. Quelltext-Editor: Zeigt einen Quelltext-Editor für Programmiersprachen, wie z.B. C#, C++ oder Delphi, an.
HTML	Entspricht der Option Web-Formulare. Gibt an, was beim Erstellen eines Web-Dokuments im Bearbeitungsfester angezeigt wird. Designer: Zeigt eine WYSIWIG-Designer an. Markup-Editor: Zeigt einen Quelltext-Editor für Auszeichnungen, wie z.B. HTML- oder ASP-Quelltext, an.

2.384 HTML-Formatierung

Tools ▶ Optionen ▶ HTML/ASP.NET-Optionen ▶ HTML-Formatierung

Verwenden Sie dieses Dialogfeld zum Festlegen von Formatierungsvorgaben für automatisch erzeugtes HTML auf der Registerseite Code.

Element	Beschreibung
Leerzeichen verwenden	Rückt erzeugtes HTML mit Leerzeichen ein.
Tabs verwenden	Rückt erzeugtes HTML mit Tabulatorzeichen ein.
Größe	Legt die Anzahl der Leerzeichen für den Einzug fest.
Ende-Tags in dieselbe Zeile setzen	Platziert schließende HTML-Tags in dieselbe Zeile wie die öffnenden.
Tags	Gibt an, ob HTML-Tags in Groß- oder Kleinbuchstaben erzeugt werden.
Attribute	Gibt an, ob HTML-Tag-Attribute in Groß- oder Kleinbuchstaben erzeugt werden.

2.385 HTML Tidy-Optionen

Tools▶**Optionen**▶**HTML/ASP.NET-Optionen**▶**HTML Tidy-Optionen**

Verwenden Sie dieses Dialogfeld, um festzulegen, wie HTML Tidy auf der Registerseite Code HTML-Code formatiert. HTML Tidy ist das Standardformatierungs-Tool der www.w3c.org.

Element	Beschreibung
HTML Tidy-Optionsfenster	Liste der HTML Tidy-Optionen, die in der IDE für die gesamte HTML-Formatierung verwendet werden können. Sie können alle Optionen über ein Kontextmenü ändern.
Beschreibung	Zeigt eine Beschreibung der markierten HTML-Tidy-Option an.

2.386 Tastaturbelegung

Tools ▶ **Optionen** ▶ **Editor-Optionen** ▶ **Tastaturbelegung**

Verwenden Sie diese Seite, um Erweiterungsmodule für Tastaturzuordnungen zu aktivieren oder zu deaktivieren und um die Reihenfolge, wie diese initialisiert werden, zu ändern.

Element	Beschreibung
Tastaturbelegungs-Module	Listet die verfügbaren Tastaturbelegungen auf. Zum Festlegen von Standard-Tastaturbelegungen verwenden Sie die Option Editor-Schnelleinstellung auf der Seite Editor-Optionen.
Erweiterungs-Module	Erweiterungsmodule sind spezielle Packages, in denen die Tastenzuordnungsfunktionen der Open-Tools-API verwendet werden. Sie werden wie normale Packages in der IDE installiert und registriert. Mit Hilfe eigener Module können neue Belegungen erstellt oder bereits definierten Zuordnungen weitere Operationen zugewiesen werden. Nach dem Installieren werden die Module in die Liste Erweiterungs-Module aufgenommen. Sie können hier mit den zugehörigen Kontrollkästchen aktiviert oder deaktiviert werden. Die Definitionen in diesen Modulen haben Vorrang vor den Zuordnungen für dieselben Tasten in der aktuellen Belegung der Liste Tastaturbelegungs-Module.
Nach oben	Verlagert das markierte Erweiterungs-Modul in der Liste eine Ebene nach oben.
Nach unten	Verlagert das markierte Erweiterungs-Modul in der Liste eine Ebene nach unten.
Strg+Alt-Tasten verwenden	Wenn die Option markiert ist, werden die Strg+Alt-Tastenkombinationen in Tastenkürzeln in der IDE verwendet. Bei nicht-markierter Option sind diese Tastenkürzel deaktiviert und Strg+Alt kann zur Ausführung anderer Funktionen, wie z.B. die Eingabe von Akzentzeichen, verwendet werden.

Siehe auch

Standard-Tastaturvorlage (siehe Seite 1335)

Tastaturvorlage IDE - Klassisch (siehe Seite 1338)

Tastaturvorlage BRIEF-Emulation (siehe Seite 1340)

Tastaturvorlage Epsilon-Emulation (siehe Seite 1342)

Tastaturvorlage Visual Studio-Emulation (siehe Seite 1344)

Tastaturvorlage Visual Basic-Emulation (siehe Seite 1345)

2.387 Sprach-Exceptions

Tools▶**Optionen**▶**Debugger-Optionen**▶**Sprach-Exceptions**

Verwenden Sie diese Seite, um zu konfigurieren, wie der Debugger ausgelöste Sprach-Exceptions behandelt. Bei unbehandelten Exceptions hält der Debugger immer an.

Element	Beschreibung
Folgende Exception-Typen ignorieren	Führt die Exception-Typen auf, die während des Testens vom Debugger ignoriert werden sollen. Markierte Typen werden ignoriert, nicht markierte werden berücksichtigt. Der Debugger unterbricht die Programmausführung nicht, wenn die ausgelöste Exception im Listenfeld aufgeführt und markiert ist. Dasselbe gilt für Exceptions, die von einer im Listenfeld aufgeführten (und dort markierten) Exception abgeleitet sind.
Hinzufügen	Zeigt das Dialogfeld Sprach-Exception hinzufügen an, in dem Sie der Liste eine benutzerdefinierte Exception hinzufügen können.
Entfernen	Entfernt eine markierte, benutzerdefinierte Exception aus der Liste. Standard-Sprach-Exceptions können nicht entfernt werden.
Bei Sprach-Exceptions benachrichtigen	Hält die Programmausführung an, sobald das Programm eine Sprach-Exception auslöst. Wenn diese Kontrollkästchen markiert ist, ignoriert der Debugger die Exception-Typen, die Sie unter Folgende Exception-Typen ignorieren ausgewählt haben. Um diesen Befehl auf Ihre Symbolleiste zu legen, damit Sie einen schneller Zugriff darauf haben, verwenden Sie die Seite Ansicht ▶ Symbolleisten ▶ Anpassen ▶ Anweisungen .

2.388 Native BS-Exceptions

[Tools](#)▶[Optionen](#)▶[Debugger-Optionen](#)▶[Native BS-Exceptions](#)

Verwenden Sie dieses Dialogfeld zum Festlegen, wie der Debugger Exceptions behandeln soll. Markieren Sie in der Liste eine Exception und wählen Sie unter Behandelt von und Beim Fortsetzen die gewünschte Option aus.

Element	Beschreibung
Exceptions	Führt die nativen Exceptions des Betriebssystems und alle benutzerdefinierten Exceptions auf.
Behandelt von	Gibt an, ob die Exception vom Debugger oder vom Programm selbst behandelt werden soll. Wenn Sie Exception-Behandlungsroutinen zu Ihrem Projekt hinzugefügt haben, wählen Sie Anwenderprogramm.
Beim Fortsetzen	Gibt an, ob die Exception weiterbehandelt oder das Projekt ohne Exception-Behandlung ausgeführt werden soll.
Hinzufügen	Zeigt das Dialogfeld Bereich für Exception hinzufügen an, in dem Sie der Liste eine benutzerdefinierte Exception hinzufügen können, die vom Debugger behandelt werden soll.
Entfernen	Entfernt eine markierte, benutzerdefinierte Exception aus der Liste. Native Betriebssystem-Exceptions können nicht entfernt werden.

2.389 Neue Tags

Tools▶ **Optionen**▶ **HTML/ASP.NET-Optionen**▶ **HTML Tidy-Optionen**▶ **Neue Tags**

Verwenden Sie diese Seite, um Tags anzugeben, die normalerweise dazu führen würden, dass HTML Tidy eine Warnung oder einen Fehler ausgibt, wie z.B. bei ASP-Tags.

Element	Beschreibung
Tags auf Blockebene	Geben Sie die Tags ein, die HTML Tidy als Block-Tags verarbeiten soll. Lassen Sie die Anfangs- und Ende-Symbole (< >) weg. Trennen Sie mehrere Tags durch ein Komma, z.B.: <code>asp:button,asp:checkbox</code>
Leere Tags	Geben Sie die Tags ein, die HTML Tidy als leere Inline-Tags verarbeiten soll. Lassen Sie die Anfangs- und Ende-Symbole (< >) weg. Trennen Sie mehrere Tags durch ein Komma.
Inline-Tags	Geben Sie die Tags ein, die HTML Tidy als nicht-leere Inline-Tags verarbeiten soll. Lassen Sie die Anfangs- und Ende-Symbole (< >) weg. Trennen Sie mehrere Tags durch ein Komma.
Pre-Tags	Geben Sie die Tags ein, die HTML Tidy in derselben Weise wie das HTML-Tag <PRE> verarbeiten soll. Lassen Sie die Anfangs- und Ende-Symbole (< >) weg. Trennen Sie mehrere Tags durch ein Komma.

2.390 Systemvariable überschreiben/Neue Anwendervariable/Anwendervariable bearbeiten

[Tools](#)▶[Optionen](#)▶[Umgebungsoptionen](#)▶[Umgebungsvariablen](#)▶[Schaltfläche Variable überschreiben, Neu und Bearbeiten](#)
oder

[Projekt](#)▶[Optionen](#)▶[Debugger](#)▶[Umgebung](#)▶[Schaltfläche Variable überschreiben, Neu und Bearbeiten](#)

Verwenden Sie dieses Dialogfeld zum Erstellen oder Bearbeiten von überschriebenen Systemvariablen.

Element	Beschreibung
Variablenname	Geben Sie einen neuen Variablennamen ein oder ändern Sie einen vorhandenen.
Variablenwert	Geben Sie einen neuen Wert ein oder ändern Sie einen vorhandenen.

2.391 Objektinspektor

Tools > Optionen > Umgebungsoptionen > Objektinspektor

Verwenden Sie diese Seite zum Konfigurieren des Objektinspektors. Auf diese Seite können Sie auch zugreifen, indem Sie im Objektinspektor mit der rechten Maustaste klicken und Eigenschaften wählen.

Element	Beschreibung
Schnelleinstellung	Zeigt eine Dropdown-Liste an, aus der Sie Einstellungen aus den folgenden Farbschemata auswählen können: Eigene Farben und Einstellungen, Vorgabe-Farben und -Einstellungen, Traditionelle Farben und Einstellungen, Klassische Farben und Einstellungen und Visual Studio(TM) Emulation.
Instanzliste anzeigen	Zeigt im oberen Bereich des Objektinspektors die Dropdown-Liste mit Komponenten und ihren Klassennamen (die sogenannte Instanzliste) an. Diese Liste ist hilfreich, wenn Sie mit vielen Komponenten oder Datenmodulen arbeiten und das gewünschte nicht sofort finden können.
Klassennamen Instanzliste	in Zeigt die Klassennamen aller Komponenten in der Instanzenliste an, nicht nur den ersten.
Statuszeile anzeigen	Zeigt im unteren Bereich des Objektinspektors die Statuszeile an. In der Statuszeile wird die Anzahl der angezeigten Eigenschaften oder Ereignisse als Ergebnis des Befehls Ansicht im Objektinspektor angegeben. Wenn alle Eigenschaften oder Ereignisse im Objektinspektor sichtbar sind, enthält die Statuszeile die Meldung Alles angezeigt.
Hintergrundraster	Fügt horizontale Hintergrundlinien ein, um die Spalten und Zeilen auf den Seiten Eigenschaften und Ereignisse sichtbar zu machen.
Integrale Höhe (nicht angedockt)	Passt den Objektinspektor so an, dass eine Zeile immer vollständig ausgefüllt wird, nicht nur teilweise, wenn Sie den Objektinspektor mithilfe des Mauszeigers in vertikaler Richtung vergrößern oder verkleinern.
Nur-Lesen-Eigenschaften anzeigen	Zeigt Komponenteneigenschaften auch dann an, wenn die Eigenschaften schreibgeschützt sind. Per Vorgabe sind sie abgedunkelt.
Nicht-Standardwerte fett	Zeigt Nicht-Standardwerte in der aktuellen Farbeinstellung für Nicht-Standardwerte in Fettschrift an.
Leiste anzeigen	Zeigt zur Verbesserung der Lesbarkeit am rechten Rand des Objektinspektors eine Leiste in der aktuellen Leistenfarbe an.
Farben	Um eines der importierten Farbschemata anzupassen, wählen Sie dieses in der Liste Schnelleinstellung aus. Wählen Sie anschließend eine Option und eine andere Farbe aus der Dropdown-Liste darunter aus. Wenn Sie beispielsweise die Farbe von Wert, die Textfarbe für Eigenschaftswerte, ändern möchten, wählen Sie Wert und klicken in der Liste Optionen auf clYellow. Mit OK können Sie die neuen Einstellungen speichern. Dabei werden jedoch nur die Änderungen an den benutzerspezifischen Farben gespeichert, nicht an den ursprünglich importierten. Um die Standardeinstellungen wiederherzustellen, klicken Sie Standard oder ein anderes Schema an.
Inline erweitern	Zeigt die Eigenschaften der referenzierten Komponente an. Klicken Sie dazu auf das Pluszeichen (+) neben der referenzierten Komponente. Per Vorgabe sind referenzierte Komponenten rot und ihre Eigenschaften grün.
Auf der Seite Ereignisse anzeigen	Zeigt die Ereignisse der referenzierten Komponente an. Per Vorgabe sind referenzierte Eigenschaften rot und ihre Ereignisse grün.

2.392 Quelloptionen

Tools > Optionen > Editor-Optionen > Quelloptionen

Verwenden Sie diese Seite zum Konfigurieren von Quelltext-Editor-Einstellungen für zahlreiche Quelltextdateitypen.

Element	Beschreibung
Quelldateityp	Wählen Sie einen vordefinierten oder angepassten Quelldateityp.
Neu	Zeigt das Dialogfeld Neuer Quelldateityp an, in dem Sie einen neuen Dateityp definieren können. Geben Sie einen Namen ein und klicken Sie auf OK; geben Sie dann in die Dropdown-Liste Erweiterungen einen Namen für die Dateinamenserweiterung ein. Sie müssen eine Erweiterung hinzufügen, wenn Sie einen neuen Quelldateityp einfügen oder die Operation abbrechen.
Löschen	Löscht einen vordefinierten oder angepassten Dateityp, der in der Dropdown-Liste Quelldateityp angezeigt wird.
Automatischer Zeileneinzug	Positioniert den Cursor beim Drücken der Taste ENTER auf das erste nicht-leere Zeichen der vorangegangenen nicht-leeren Zeile im Quelltext-Editor.
Einzug mit Tab	Fügt beim Drücken der Taste TAB ein Tab-Zeichen in den Quelltext-Editor ein. Falls deaktiviert, fügt die Option Leerzeichen statt Tabulatorzeichen ein. Wenn Automatische Tabs aktiviert sind, ist diese Option ausgeschaltet. Um Tab-Zeichen sichtbar zu machen, wählen Sie Tab-Zeichen anzeigen.
Automatische Tabs	Geht zu dem ersten nicht-weißen Zeichen in der vorangegangenen Zeile. Wenn Einzug mit Tab aktiviert ist, ist diese Option ausgeschaltet.
Cursor durch Tabs	Aktiviert die Pfeiltasten, um den Cursor an den Anfang jedes Tabulators zu setzen.
Füllen mit Tabs	Beginnt jede automatisch eingezogene Zeile mit einer möglichst geringen Anzahl an Zeichen und verwendet dafür Tabulatoren und Leerzeichen.
Rücktaste löscht Einzug	Bringt die Einfügemarke an die vorherige Einzugsebene (rückt sie aus), wenn Sie die Taste RÜCK betätigen, und falls der Cursor auf dem ersten nicht-leeren Zeichen einer Zeile steht.
Leerzeichen beibehalten	Verhindert, dass nachgestellte Leerzeichen gelöscht werden.
Tab-Zeichen anzeigen	Zeigt Tabulatorzeichen als >> an, falls Einzug mit Tab ausgewählt ist.
Leerzeichen anzeigen	Zeigt eingegebene Leerzeichen als Punkte (.) an.
Syntaxhervorhebung benutzen	Aktiviert die Syntaxhervorhebung. Um die Vorgaben für die Syntaxhervorhebung einzustellen, verwenden Sie die Optionen der Seite Farbe.
Zeigt Zeilenumbrüche an	Zeigt Zeilenumbruchssymbole am Ende jeder Zeile an.
Aktuelle Zeile markieren	Markiert die aktuelle Zeile im Quelltext-Editor.
Syntaxhervorhebung	Wählen Sie eine Option, wenn Sie das Anzeigeformat von Quelltextelementen ändern möchten. Diese Option ist erst verfügbar, nachdem die Option Syntaxhervorhebung benutzen ausgewählt wurde.
Einrückung	Legt die Anzahl von Leerzeichen fest, um die ein markierter Block eingerückt wird. Die Vorgabe lautet 2, und die obere Grenze ist 16.
Tabstops	Legen Sie die Tabulatorgröße fest, um die der Cursor beim Drücken der Taste TAB vorgerückt wird. Geben Sie einen oder mehrere durch Leerzeichen getrennte Integer-Werte ein. Bei mehreren Angaben bestimmen die Zahlen die Spalte, an der die Tabulatorzeichen gesetzt werden. Jeder folgende Wert muss größer als sein Vorgänger sein. Wird nur ein Tabulatorzeichen angegeben, entspricht die Zahl der Anzahl an Leerstellen, um die der Cursor vorgerückt wird.

2.393 Farben

Tools▶**Optionen**▶**Umgebungsoptionen**▶**Tool-Palette**▶**Farben**

Verwenden Sie dieses Dialogfeld, um die Farben der Tool-Palette zu ändern.

Element	Beschreibung
Farbschemata	Führt vordefinierte Farbkombinationen auf. Wählen Sie aus der Dropdown-Liste ein Farbschema aus. Es wird sofort für die Tool-Palette übernommen. Die vordefinierten Farbschemata lassen sich nicht verändern. Sie können aber ein Farbschema auswählen und dann die zugehörigen Farben beliebig verändern, um ein eigenes, unbenanntes Farbschema zu erstellen.
Basis (Kategoriefarben)	Legt die Farbe fest, die für den Hintergrund des Kategorienfensters verwendet wird.
Verlauf (Kategoriefarben)	Legt die Farbe fest, die für die Schattierung der Basisfarbe verwendet wird.
Text	Legt die Farbe fest, die für die Kategorietitel verwendet wird.
Titel fett	Formatiert die Kategorietitel in Fettschrift.
Vertikaler Verlauf	Übernimmt die Verlaufsfarbe für den oberen Bereich des Kategorienfensters anstatt für die linke Seite des Fensters.
Rahmen nur bei Titel	Übernimmt die Basisfarbe, Verlaufsfarbe und Textfarbe nur für die Kategorietitel, nicht für das gesamte Kategorienfenster.
+/- Symbole verwenden	Zeigt anstatt der Caret-Zeichen die Zeichen Plus (+) und Minus (-) neben den Kategorietiteln an.
Vertikale Titel	Zeigt die Kategorietitel vertikal auf der linken Seite des Kategorienfensters an.
Normal	Legt die Farbe fest, die für den Hintergrund von Schaltflächen verwendet wird.
Markiert	Legt die Farbe fest, die für den Hintergrund von ausgewählten Schaltflächen verwendet wird.
Aktiv	Legt die Farbe fest, die für Schaltflächen verwendet wird, wenn der Mauszeiger über die Schaltfläche bewegt wird.
Basis (Hintergrundfarben)	Legt die Farbe fest, die für den Rahmen um die einzelnen Kategorienfenster verwendet wird.
Verlauf (Hintergrundfarben)	Legt die Farbe fest, die für den Schatten des Rahmens um die einzelnen Kategorienfenster verwendet wird.
Verlaufsrichtung	Legt fest, ob der mit der Verlaufsfarbe dargestellte Schatten um das Kategorienfenster vertikal oder horizontal verläuft.

Tip: Die Auswirkungen der in diesem Dialogfeld vorgenommen Änderungen werden sofort in der Tool-Palette angezeigt, ohne dass Sie das Dialogfeld schließen müssen.

Siehe auch

Der Tool-Palette Komponenten hinzufügen (↗ siehe Seite 70),

2.394 Tool-Palette

Tools▶**Optionen**▶**Umgebungsoptionen**▶**Tool-Palette**

Verwenden Sie dieses Dialogfeld, um das Erscheinungsbild der Tool-Palette zu ändern.

Element	Beschreibung
Schaltflächengröße	Ändert die Größe der Symbole, die zur Darstellung der Elemente in der Tool-Palette angezeigt werden.
Kategorien automatisch ausblenden	Legt fest, dass jeweils nur eine Kategorie erweitert dargestellt wird.
Vertikale Kategorietitel	Zeigt die Kategorietitel vertikal links neben den Elementen in der Tool-Palette an.
Neuanordnung der Palette sperren	Deaktiviert in der Tool-Palette die Neuanordnung der Elemente durch Drag&Drop.
Schaltflächentitel anzeigen	Zeigt die Titel neben den Elementsymbolen an.
Vertikales Layout	Zeigt die Kategorien vertikal an.
Paletten-Experten anzeigen	Zeigt die Elemente aus dem Dialogfeld Objektgalerie in der Tool-Palette an, wenn der Quelltext-Editor aktiv ist oder die Projektverwaltung den Fokus hat. Das Dialogfeld Objektgalerie kann auch über den Menübefehl Datei ▶ Neu ▶ Weitere geöffnet werden.
Code-Snippets anzeigen	Zeigt in der Tool-Palette Codefragmente an, wenn der Quelltext-Editor aktiv ist.
Designer-Elemente immer anzeigen	Zeigt Designer-Elemente an, auch wenn der Quelltext-Editor aktiv ist (entspricht dem Verhalten von Delphi 7). Deaktivieren Sie diese Option, wenn bei aktivem Quelltext-Editor in der Tool-Palette keine Designer-Elemente angezeigt werden sollen.

Tip: Die Auswirkungen der in diesem Dialogfeld vorgenommen Änderungen werden sofort in der Tool-Palette angezeigt, ohne dass Sie das Dialogfeld schließen müssen.

Siehe auch

Der Tool-Palette Komponenten hinzufügen (☞ siehe Seite 70)

Komponenten in ein Formular einfügen (☞ siehe Seite 55)

Code-Snippets verwenden (☞ siehe Seite 50)

2.395 Farbe

Tools▸**Optionen**▸**Optionen für Übersetzungs-Tools**▸**Farben**

Verwenden Sie dieses Dialogfeld zum Definieren des Farbschemas für den Translation-Manager.

Element	Beschreibung
Farbschema	Ermöglicht die Auswahl eines vordefinierten Farbschemas.
Speichern unter	Fügt Ihr eigenes Farbschema der Liste hinzu.
Entfernen	Entfernt ein benutzerdefiniertes Farbschema aus der Liste.
Element	Führt die Elemente des Translation-Managers auf. Um für ein Element eine Farbe festzulegen, klicken Sie auf das Element und dann auf ein Farbfeld.
Beispiel	Zeigt, wie einige der gewählten Farben im Translation-Manager aussehen.
Farben verwenden	Steuert die Farbcodierung im Translation-Manager.

Siehe auch

Anwendungen lokalisieren ( siehe Seite 1437)

2.396 Schrift

Tools▶**Optionen**▶**Optionen für Übersetzungs-Tools**▶**Schrift**

Verwenden Sie dieses Dialogfeld zum Festlegen der Schriftart für den Translation-Manager.

Element	Beschreibung
Tabellenschriften	Zeigt die in Ihrem Projekt verfügbaren Sprachen und Schriftarten an.
Beispiel	Enthält ein Beispiel für die gewählte Schriftart.
Schrift	Zeigt das Dialogfeld Schriftart an, in dem Sie eine Schriftart, einen Schriftschnitt, eine Schriftgrad, einen Effekt, eine Farbe und ein Skript für die unter Tabellenschriften gewählte(n) Sprache(n) auswählen können.

Siehe auch

Anwendungen lokalisieren (siehe Seite 1437)

2.397 Formular-Designer

Tools▸**Optionen**▸**Optionen für Übersetzungs-Tools**▸**Formular-Designer**

Verwenden Sie dieses Dialogfeld zum Festlegen von Präferenzen für die Formulare, die im Translation-Manager angezeigt werden.

Element	Beschreibung
Raster anzeigen	Zeigt das Raster im Formular mit Punkten an.
Am Raster ausrichten	Richtet die Komponenten auf dem Formular automatisch an der nächsten Rasterlinie aus. Es ist nicht möglich, Komponenten zwischen den Rasterlinien zu platzieren.
Rastergröße	Setzt den Pixel-Abstand zwischen den Rasterlinien entlang der X- und Y-Achse. Legen Sie eine Zahl (zwischen 2 und 128) fest, um den Abstand zu erhöhen.

2.398 Packages

Tools▶**Optionen**▶**Packages**

Verwenden Sie dieses Dialogfeld, um dem externen Translation-Manager Entwurfszeit-Packages aus dem Ressourcenprojekt hinzuzufügen oder daraus zu entfernen.

Element	Beschreibung
Hinzufügen	Ermöglicht das Suchen von Entwurfszeit-Packages, die Sie Ihrem Projekt hinzufügen möchten. Sie können mehrere Packages auf einmal hinzufügen.
Entfernen	Entfernt die markierten Entwurfszeit-Packages aus dem Projekt.

2.399 Optionen für Übersetzungs-Tools

Tools > Optionen > Optionen für Übersetzungs-Tools

Verwenden Sie dieses Dialogfeld zum Konfigurieren des Satellite-Assemblierungs-Experten, des Ressourcen-DLL-Experten, des Translation-Managers und des Übersetzungswörterbuchs.

Element	Beschreibung
Wörterbuch automatisch abfragen	Füllt Ressourcenmodule bei jeder Aktualisierung der Assemblierungen automatisch mit Übersetzungen für alle Strings, für die im Übersetzungswörterbuch Übereinstimmungen vorhanden sind. Wenn im Übersetzungswörterbuch nur eine Übereinstimmung für einen String gefunden wird, wird dieser übersetzte String in das Ressourcenmodulprojekt kopiert. Wenn mehrere Übereinstimmungen vorhanden sind, wird die erste Übersetzung im Wörterbuch in die Assemblierung kopiert. Sie können dieses Verhalten ändern, indem Sie Tools > Optionen > Optionen für Übersetzungs-Tools wählen, das Register Wörterbuch anklicken, und dann die Einstellung von Aktion bei mehreren Vorkommen ändern.
Projekte automatisch compilieren	Compiliert Projekte ohne vorherige Rückfrage, wenn dies für Tools der Übersetzungsumgebung erforderlich ist (z.B. beim Ausführen des Satellite-Assemblierungs-Experten).
Translation-Manager nach RDW anzeigen	Öffnet den Translation-Manager automatisch nach dem Ausführen des Satellite-Assemblierungs-Experten oder des Ressourcen-DLL-Experten, wenn kein Ressourcenmodulsprojekt in der Projektverwaltung aktiv war, als Sie den Experten aufgerufen haben.
Anführungszeichen automatisch einfügen	Schließt die übersetzten Strings in die erforderlichen Anführungszeichen ein, wenn die Strings nicht bereits Apostrophe, Anführungszeichen oder Steuerzeichen (wie z.B. #13) enthalten.
Status "Neu übersetzt" verwenden	Wenn ein String manuell oder automatisch aus dem Übersetzungswörterbuch übersetzt wird, wird der Status des String in Neu übersetzt anstatt in Übersetzt geändert. Anhand dieses Status können die Einträge bestimmt werden, die in der aktuellen Übersetzung übersetzt wurden.
Dateien automatisch speichern	Speichert das aktuelle Projekt ohne Rückfrage automatisch (z.B. vor dem Schließen des Managers oder vor dem Ausführen des Satellite-Assemblierungs-Experten oder des Ressourcen-Modul-Experten).
Leere Einträge ausblenden	Blendet Dateien in der Hierarchiedarstellung des Registers Arbeitsbereich des Translation-Managers aus, die keine übersetzungsrelevanten Einträge enthalten, wie z.B. .nfn- und .resN-Dateien. Dadurch kann bei der Verarbeitung von mehreren Dateien, die teilweise leer sind, mit den Befehlen Strings dem Wörterbuch hinzufügen oder Strings aus dem Wörterbuch holen die Verarbeitungsgeschwindigkeit erhöht werden.
Externer Editor	Gibt den Namen des Editors an, der als externer Editor im Translation-Manager verwendet werden soll, z.B. <code>notepad.exe</code> .

Siehe auch

Anwendungen lokalisieren (siehe Seite 1437)

2.400 Wörterbuch

Tools▶**Optionen**▶**Optionen für Übersetzungs-Tools**▶**Wörterbuch**

Verwenden Sie dieses Dialogfeld zum Konfigurieren des Übersetzungswörterbuchs.

Element	Beschreibung
Dateiname	Legt fest, wo sich das Wörterbuch befindet. Das Wörterbuch ist eine Datenbank für Übersetzungen, die in verschiedenen Projekten gemeinsam genutzt werden kann. Geben Sie den vollständigen Pfad der .tmx -Datei an, in der das Übersetzungswörterbuch gespeichert ist. Standardangabe: \$(ETM)\default.tmx
Aktion bei Doppelten	Legt fest, welche Aktion ausgeführt wird, wenn für einen String zwei Übersetzungen im Wörterbuch gefunden werden. Überspringen fügt den String nicht hinzu. Hinzufügen fügt den String dem Wörterbuch hinzu, wenn kein übersetzter String für den Original-String vorhanden ist. Immer hinzufügen fügt den String dem Wörterbuch immer hinzu, auch wenn der String bereits im Wörterbuch enthalten ist. Ersetzen überschreibt den vorhandenen String mit dem neuen String. Auswahl anzeigen zeigt ein Dialogfeld mit Auswahlmöglichkeiten an.
Aktion bei mehreren Vorkommen	Legt fest, welche Aktion ausgeführt wird, wenn für einen String mehrere Übersetzungen im Wörterbuch gefunden werden. Überspringen ruft keine Übersetzungen ab, wenn es mehrere Übereinstimmungen gibt. Ersten verwenden übernimmt die erste Übereinstimmung. Auswahl anzeigen zeigt ein Dialogfeld mit Auswahlmöglichkeiten an.

Siehe auch

Anwendungen lokalisieren (siehe Seite 1437)

2.401 Übersetzungswörterbuch

Ansicht▸Translation-Manager▸Übersetzungswörterbuch

Verwenden Sie dieses Dialogfeld zum Suchen, Bearbeiten und Löschen von Ressourcen-Strings. Wenn der Translation-Manager aktiv ist, können Sie übersetzte Strings im Übersetzungswörterbuch speichern und daraus abrufen. Per Vorgabe speichert das Übersetzungswörterbuch Daten in der Datei `default.tmx` im Verzeichnis RAD Studio `/bin`.

Mithilfe der Symboleistenschalter können Sie ein Übersetzungswörterbuch (`.tmx`-Datei) erstellen, öffnen und speichern. Nach dem Öffnen einer `.tmx`-Datei lassen sich mit den Befehlen des Kontextmenüs Aktionen für einzelne Ressourcen-Strings ausführen.

Tip: Um das Übersetzungswörterbuch zu konfigurieren, schließen Sie es, wählen **Tools**▸**Optionen**▸**Optionen für Übersetzungs-Tools**▸**Wörterbuch**.

Siehe auch

Anwendungen lokalisieren (☞ siehe Seite 1437)

Sprachen zu einem Projekt hinzufügen (☞ siehe Seite 86)

Ressourcendateien im Translation-Manager bearbeiten (☞ siehe Seite 88)

Externen Translation-Manager einrichten (☞ siehe Seite 91)

2.402 Typbibliothek (Delphi)

Tools▶**Optionen**▶**Umgebungsoptionen**▶**Delphi-Optionen**▶**Typbibliothek**

Verwenden Sie dieses Dialogfeld, um Optionen für den Typbibliothek-Editor festzulegen.

Element	Beschreibung
SafeCall-Funktionsumwandlung	<p>Legt fest, welche Funktionen als safecall deklariert werden, wenn in Delphi angegebene Deklarationen in der generierten Typbibliothek in IDL (Interface Definition Language) umgewandelt werden.</p> <p>Safecall-Funktionen implementieren automatisch COM-Konventionen zur Fehler- und Exception-Behandlung, indem HRESULT-Fehlercodes in Exceptions umgewandelt werden. Wenn Sie Funktionsdeklarationen in IDL eingeben, müssen Sie als Aufrufkonvention explizit safecall oder stdcall angeben.</p> <p>Alle V-Table-Interfaces verwendet SafeCall für alle Interfaces.</p> <p>Nur duale Interfaces verwendet SafeCall nur für duale Interfaces.</p> <p>Nicht umwandeln verwendet die SafeCall-Aufrufkonvention nicht.</p>
Pascal	Sprache von Delphi. Sie werden vermutlich Delphi für CORBA-Schnittstellen benutzen, weil sich CORBA IDL leicht von Microsoft IDL unterscheidet.
IDL	Microsoft Interface Definition Language
Spezielle CoClass-Flags beim Importieren ignorieren	Beim Importieren eines ActiveX-Steuerelements werden nur CoClasses in die Typbibliothek übernommen, die nicht als Hidden, Restricted oder Predefined, sondern als Can Create (eigentlich <i>noncreatable</i>) markiert sind. Diese Flags sollten gesetzt werden, wenn das Objekt für allgemeine Zwecke verwendet wird. Wenn Sie allerdings ein Steuerelement nur für eine interne Anwendung erstellen möchten, können Sie die Flags überschreiben, um CoClass-Wrappers zu generieren. In diesem Fall würden Sie für Spezielle CoClass-Flags beim Importieren ignorieren Hidden und Restricted markieren sowie Can Create (<i>noncreatable</i>) deaktivieren.
Predefined	Client-Anwendungen sollten automatisch eine einzelne Instanz dieses Objekts erzeugen.
Restricted	Eine als Restricted gekennzeichnete CoClass wird von Tools, die auf COM-Objekte zugreifen, ignoriert. Sie wird von der Typbibliothek zur Verfügung gestellt, der Zugriff muss jedoch autorisiert sein.
Verborgen	Das Interface existiert, sollte aber in einem benutzerorientierten Browser nicht angezeigt werden.
Can Create	Das Interface kann mit <i>CoCreateInstance</i> erzeugt werden.
Änderungen vor Aktualisierung anzeigen	<p>Zeigt das Dialogfeld Aktualisierung durchführen an, in dem Sie vorgeschlagene Änderungen für das Aktualisieren, Speichern oder Registrieren der Typbibliothek überprüfen können.</p> <p>Ist diese Option nicht markiert, dann aktualisiert der Editor für Typbibliotheken automatisch die Quellen des zugehörigen Objekts, wenn Sie Änderungen im Editor vornehmen.</p>

2.403 WebSnap

Tools > Optionen > WebSnap

Auf dieser Seite nehmen Sie die WebSnap-Einstellungen vor.

Element	Beschreibung
Debugger aktivieren	Aktiviert den Script-Debugger, wenn ein Fehler beim Debuggen eines Webseiten-Moduls auftritt.
HTML-Dateierweiterung	Gibt die Dateierweiterung an, die der Experte Neue WebSnap-Anwendung den HTML-Dateien zuweist, die er generiert.
Beispiel-Bilddatei	Diese Datei wird von Adapterkomponenten verwendet, um ein Beispielbild für den Fall anzuzeigen, dass das richtige Bild während des Entwurfs noch nicht zur Verfügung steht. Klicken Sie die Schaltfläche Durchsuchen an, um den Pfad für das Beispielbild festzulegen.

2.404 Windows Forms-Designer

Tools▶**Optionen**▶**Umgebungsoptionen**▶**Windows Forms-Designer**

Verwenden Sie dieses Dialogfeld zum Festlegen von Vorgaben für Windows Forms.

Element	Beschreibung
Raster anzeigen	Zeigt das Raster im Formular mit Punkten an. Das Ändern dieser Einstellung wirkt sich nur auf Formulare aus, die <i>nach</i> dieser Änderung erstellt werden. Um diese Einstellung für ein vorhandenes Formular zu ändern, müssen Sie die Eigenschaft DrawGrid im Objektinspektor ändern.
Am Raster ausrichten	Richtet die Komponenten auf dem Formular automatisch an der nächsten Rasterlinie aus. Es ist nicht möglich, Komponenten zwischen den Rasterlinien zu platzieren. Das Ändern dieser Einstellung wirkt sich nur auf Formulare aus, die <i>nach</i> dieser Änderung erstellt werden. Um diese Einstellung für ein vorhandenes Formular zu ändern, müssen Sie die Eigenschaft SnapToGrid im Objektinspektor ändern.
Rastergröße	Setzt den Pixel-Abstand zwischen den Rasterlinien entlang der X- und Y-Achse. Legen Sie eine Zahl (zwischen 2 und 128) fest, um den Abstand zu erhöhen. Das Ändern dieser Einstellung wirkt sich nur auf Formulare aus, die <i>nach</i> dieser Änderung erstellt werden. Um diese Einstellung für ein vorhandenes Formular zu ändern, müssen Sie die Eigenschaft GridSize im Objektinspektor ändern.
Standardstil	Zeigt bei neu erzeugtem Code Klammern in derselben Zeile wie Funktionsdeklarationen an (vorhandene Klammern werden nicht geändert).
C-Stil	Zeigt bei neu erzeugtem Code Klammern in der Zeile nach einer Funktionsdeklaration an (vorhandene Klammern werden nicht geändert).

2.405 Tools-Eigenschaften

Tools > Tools konfigurieren > Schaltfläche Hinzufügen oder Bearbeiten

Verwenden Sie dieses Dialogfeld, um Eigenschaften eines im Menü Tools aufgeführten Programms einzugeben oder zu bearbeiten.

Element	Beschreibung
Titel	Geben Sie einen Namen für das hinzuzufügende Programm ein. Dieser Name wird im Menü Tools angezeigt. Sie können ein Tastenkürzel für den Menübefehl festlegen, indem Sie dem Buchstaben ein Und-Zeichen (&) voranstellen. Wenn Sie ein bereits vorhandenes Tastenkürzel verwenden, wird im Dialogfeld Tools-Optionen ein rotes Sternchen (*) neben den Programmnamen angezeigt.
Programm	Geben Sie die Position für das hinzuzufügende Programm an. Sie können den vollständigen Programmpfad angeben. Um Ihre Laufwerke und Verzeichnisse nach dem Pfad- und Dateinamen des Programms zu durchsuchen, klicken Sie die Schaltfläche Durchsuchen an.
Arbeitsverzeichnis	Geben Sie das Arbeitsverzeichnis für das hinzuzufügende Programm an. Es wird ein Standardarbeitsverzeichnis festgelegt, wenn Sie den Programmnamen in dem Textfeld Programm auswählen. Sie können den Verzeichnispfad bei Bedarf ändern.
Parameter	Geben Sie die Parameter ein, die beim Start an das Programm übergeben werden. Beispielsweise ist es möglich, beim Programmstart einen Dateinamen zu übergeben. Tippen Sie die Parameter ein oder verwenden Sie die Schaltfläche Makros, um Startparameter festzulegen. Sie können mehrere Parameter und Makros angeben.
Makros	Erweitert das Dialogfeld Tools-Eigenschaften und zeigt eine Liste der verfügbaren Makros an. Sie können diese Makros verwenden, um die Startparameter für Ihre Anwendung bereitzustellen. Markieren Sie ein Makro und klicken Sie Einfügen an, um das Makro in das Textfeld Parameter zu übernehmen.
Durchsuchen	Zeigt das Dialogfeld Tools auswählen an, in dem Sie zu einem Programm navigieren können.

2.406 XML-Mapper

Tools > XML-Mapper

Verwenden Sie das XML-Mapper, um während des Entwurfs Datenpakete, die von Client-Datenmengen verwendet werden, generischen XML-Dokumenten und umgekehrt zuzuordnen. Jede Zuordnung beschreibt die Entsprechungen zwischen den Knoten eines XML-Dokuments und den Feldern in einem Datenpaket.

Sie können Zuordnungen von einem vorhandenen XML-Schema (oder -Dokument) zu einer Client-Datenmenge, die Sie definieren, von einem vorhandenen Datenpaket zu einem neuen XML-Schema, das Sie definieren, oder zwischen einem vorhandenen XML-Schema und einem vorhandenen Datenpaket definieren.

Seite Dokumentansicht

Diese Seite zeigt den Inhalt des aktuell geladenen XML-Dokuments als hierarchische Baumstruktur. Jeder Knoten der Hierarchie repräsentiert ein Tag oder ein Tag-Attribut in dem XML-Dokument. Neben jedem Knoten befindet sich ein Symbol, das den Typ des Tags angibt.

Element	Beschreibung
	Repräsentiert einen Elementknoten. Das ist ein Tag, das anderen Knoten (Tags) übergeordnet ist, aber keinen Wert besitzt. Der Name des Knotens ist der Tag-Name. Typischerweise werden Elementknoten Datenmengen zugeordnet (dem Datenpaket selbst oder einer verschachtelten Detailmenge), obwohl sie auch Feldern zugeordnet werden können, deren Werte aus den Werten der untergeordneten Knoten zusammengesetzt werden.
	Repräsentiert einen Textknoten. Textknoten repräsentieren getaggte Elemente mit Textwerten. In der Hierarchie haben sie die Form <i>Knotenname="TextWert"</i> , wobei <i>Knotenname</i> der Tag-Name und <i>TextWert</i> der Text ist, der zwischen dem öffnenden und dem schließenden Tag steht. Typischerweise werden Textknoten Feldern im Datenpaket zugeordnet.
	Repräsentiert einen Attributknoten. Attributknoten entsprechen den Attributen der Tags des übergeordneten Elements in dem XML-Dokument. In der Hierarchie haben Sie die Form <i>Knotenname="AttributWert"</i> , wobei <i>Knotenname</i> der Name des Attributs und <i>AttributWert</i> dessen Wert ist. Typischerweise werden Attributknoten Feldern im Datenpaket zugeordnet, wobei das Element, dem sie als Attribut dienen, einem Datensatz zugeordnet ist.
	Repräsentiert einen verschachtelten Knoten. Verschachtelte Knoten sind Elementknoten, die fortlaufend in einem XML-Dokument wiederholt werden können. Typischerweise werden verschachtelte Knoten Datensätzen im Datenpaket zugeordnet.
Datenansicht	<p>Ist diese Option nicht markiert, werden nur die Namen und Typen der Knoten angezeigt. Es werden für Text- oder Attributknoten keine Werte angezeigt und für jeden verschachtelten Knoten wird nur eine einzelne Instanz angezeigt.</p> <p>Ist diese Option markiert, werden für Text- oder Attributknoten Beispielwerte und Wiederholungen für verschachtelte Knoten angezeigt. Wenn Sie eine XML-Beispieldatei laden, zeigt Datenansicht die in dieser Datei gespeicherten Werte. Wenn das Dokument aus einem Schema oder Datenpaket generiert wurde, werden Beispielwerte für die Knoten erzeugt.</p> <p>Bei großen XML-Dokumenten ist es manchmal einfacher die Option Datenansicht zu deaktivieren, weil ohne die Einzelheiten die logische Struktur deutlicher wird.</p>

Seite Schema-Ansicht

Diese Seite zeigt die XML-Schemainformationen. Die Seite enthält drei Registerkarten, die die verschiedenen vom XML-Zuordnungs-Tool unterstützten Schemaformate repräsentieren. Die Formate sind DTD, XDR (Reduced XMLData) und XSD (XML-Schema). Die Informationen auf der Seite Schema-Ansicht können aus einer Datei eingelesen oder aus der aktuellen XML-Dokument erschlossen werden.

Seite Knoteneigenschaften

Auf dieser Seite können Sie dem im XML-Dokument markierten Knoten Eigenschaften zuweisen. Mit diesen Eigenschaften wird bei der Erzeugung einer Umwandlungsdatei sicher gestellt, dass aus XML-Dokumenten erzeugte Datenpakete die korrekten Feldtypen und aus Datenpaketen erzeugte XML-Dokumente die korrekten Knoten besitzen. Die Umwandlungsdatei enthält die Werte, auf der Seite Knoteneigenschaften aktuell festgelegt sind.

Element	Beschreibung
UTF-8 codiert	Steuert, ob Zeichen des erweiterten Zeichensatzes mit UFT-8 (wenn markiert) oder mit einer Escape-Sequenz (wenn nicht markiert) im XML codiert werden. Wenn die Option markiert ist, ändert sich die Eigenschaft <i>Data Format</i> für Strings, Memos und WideStrings von ANSI in UTF-8.
Benutzerdefinierte Übersetzung	Steuert, ob der markierte Knoten automatisch umgewandelt werden soll. Diese Option ermöglicht Umwandlungen, die über einfache 1:1-Zuordnungen hinausgehen, und für die ein Datentyp angegeben werden kann. Sie können beispielsweise einen benutzerdefinierten Knoten anlegen, um einen Elementknoten mit untergeordneten Knoten für Vorname und Nachname in ein einziges "Namen"-Feld in dem Datenpaket zu konvertieren. Wenn Sie die Option Benutzerdefinierte Übersetzung aktivieren, müssen Sie zur Repräsentation des Knotens einen ID-String zuweisen. Dieser ID-String wird an die Ereignisbehandlungsroutine <i>OnTranslate</i> von <i>TXMLTransform</i> übergeben, damit die Übersetzung im Code durchgeführt werden kann. Wenn ein Knoten nicht als benutzerdefiniert gekennzeichnet wird, wird das Ereignis <i>OnTranslate</i> für diesen Knoten nicht ausgelöst.
Knotenbeschreibung	Optional. Geben Sie eine Beschreibung für den Knoten ein. Diese Beschreibung wird weder dem XML-Dokument noch dem Datenpaket hinzugefügt, erleichtert aber die Identifizierung des Zwecks einer Basiselementmenge, wenn Sie Eigenschaftsmengen in einer Knotenablage datei speichern.

Tip: Um die aktuellen Knoteneinstellungen in einer Knotenablage datei zu speichern, klicken Sie mit der rechten Maustaste und wählen Ablage speichern. Um die aktuellen Knoteneinstellungen aus einer Knotenablage datei zu laden, klicken Sie mit der rechten Maustaste und wählen Ablage öffnen. Um alle auf der Seite Knoteneigenschaften vorgenommenen Änderungen rückgängig zu machen und zu den Werten zurückzukehren, die aus dem XML-Dokument erschlossen wurden, klicken Sie mit der rechten Maustaste und wählen Entfernen.

Seite Zuordnung

Auf dieser Seite lassen sich Zuordnungen zwischen Feldern in dem Datenpaket und Knoten in dem XML-Dokument festlegen, eine Umwandlungsdatei erstellen und speichern.

Im oberen Bereich der Seite wird eine zweispaltige Tabelle angezeigt, die die Knoten aus dem XML-Dokument und die entsprechenden Felder aus dem Datenpaket enthält. Bei der ersten Anzeige der Seite Zuordnung ist die Tabelle leer. Um eine Zuordnung zu definieren, muss die Tabelle gefüllt werden.

Anmerkung: Es können nur Knoten mit Werten (Text- und Attributknoten) oder Knoten, die auf der Seite Knoteneigenschaften als benutzerdefiniert gekennzeichnet wurden, hinzugefügt werden.

Seite Felderansicht

Diese Seite zeigt die Feldattribute für alle Felder in dem Datenpaket an. Jeder Knoten in der Hierarchie repräsentiert eine Datenmenge, ein Feld oder ein Feldattribut.

Element	Beschreibung
	Repräsentiert das gesamte Datenpaket oder ein Datenmengenfeld. Die einem Datenmengenfeld untergeordneten Knoten repräsentieren die Felder in dieser Datenmenge.
	Repräsentiert ein Feld, das kein Datenmengenfeld ist. Die einem Feldknoten untergeordneten Knoten repräsentieren die Attribute des Feldes.

•	Repräsentiert ein Feldattribut, wie Datentyp, Maximallänge usw. Der Knoten trägt eine Bezeichnung in der Form <i>AttributName</i> = <i>Wert</i> , wobei <i>AttributName</i> der Name des Attributs und <i>Wert</i> dessen Wert ist.
---	---

2

Seite Datenpaketansicht

Diese Seite zeigt die Struktur des Datenpaketes an. Die Symbole in dieser Ansicht entsprechen denjenigen in der Dokumentansicht, weil Datenpaket als spezielle Typen von XML-Dokumenten behandelt werden können.

Element	Beschreibung
	Repräsentiert einen Elementknoten. Elementknoten in Datenpaketen repräsentieren Datenmengen oder Datenmengenfelder.
	Repräsentiert einen Attributknoten. Attributknoten in Datenpaketen repräsentieren Felder (keine Datenmengenfelder).
	Repräsentiert einen verschachtelten Knoten. Verschachtelte Knoten in Datenpaketen repräsentieren Datensätze.

Anmerkung: Das XML-Zuordnungs-Tool kann sowohl binäre Datenpakete (`.cds`-Dateien) als auch Datenpakete, die als XML gespeichert wurden, verwenden. Wenn Sie ein Datenpaket im binären Format verwenden, wird es von dem XML-Zuordnungs-Tool in das XML-Format konvertiert.

Siehe auch

[XML in Datenbankanwendungen verwenden](#)

2.407 Der Objektablage hinzufügen

Verwenden Sie dieses Dialogfeld, um Strings in der ausgewählten Unit dem Übersetzungswörterbuch hinzuzufügen. Dieses Dialogfeld wird angezeigt, wenn Sie auf der Registerseite Arbeitsbereich des Translation-Managers einen Knoten mit der rechten Maustaste anklicken und den Befehl Strings dem Wörterbuch hinzufügen wählen.

Um einzelne Strings und nicht Strings für eine gesamte Unit hinzuzufügen, klicken Sie den String im Translation-Manager mit der rechten Maustaste an und wählen **Wörterbuch**►**Strings dem Wörterbuch hinzufügen**.

Die folgenden Optionen legen die Kriterien für das Hinzufügen von Strings fest.

Element	Beschreibung
Status	Fügt Strings basierend des in der Spalte Status der Registerseite Arbeitsbereich angezeigten Status ein. Wählen Sie den hinzuzufügenden Status aus.
Aktion bei Doppelten	Legt fest, welche Aktion ausgeführt wird, wenn für einen String zwei Übersetzungen im Wörterbuch gefunden werden. Überspringen fügt den String nicht hinzu. Hinzufügen fügt den String dem Wörterbuch hinzu, wenn kein übersetzter String für den Original-String vorhanden ist. Immer hinzufügen fügt den String dem Wörterbuch immer hinzu, auch wenn der String bereits im Wörterbuch enthalten ist. Ersetzen überschreibt den vorhandenen String mit dem neuen String. Auswahl anzeigen zeigt ein Dialogfeld mit Auswahlmöglichkeiten an.
Mit Kontextinformationen	Fügt den Unit-Pfad und den in der Spalte Id der Registerseite Arbeitsbereich angezeigten Wert in das Übersetzungswörterbuch ein. Diese Kontextinformationen werden in der Statuszeile angezeigt, wenn ein String im Übersetzungswörterbuch markiert ist.
Wert	Legt fest, ob ein String basierend auf Änderungen des Originalwertes hinzugefügt wird. Geändert fügt den String nur dann ein, wenn der Original- und der übersetzte Wert unterschiedlich sind. Unverändert fügt den String auch dann ein, wenn der Original- und der übersetzte Wert gleich sind. Keine Überprüfung fügt den String ungeachtet dessen ein, ob er verändert wurde oder nicht, solange er den anderen in diesem Dialogfeld festgelegten Kriterien entspricht.
Kommentar	Fügt Strings basierend des in der Spalte Kommentar der Registerseite Arbeitsbereich angezeigten Textes ein oder schließt sie aus. Geben Sie den Kommentartext in das Eingabefeld ein und markieren Sie Einbeziehen, um Strings mit einem übereinstimmenden Kommentar hinzuzufügen, oder markieren Sie Nicht einbeziehen, um diese Strings zu übergehen.

Tip: Zum Setzen von allgemeinen Optionen für das Übersetzungswörterbuch wählen Sie **Tools**►**Optionen**►**Optionen für Übersetzungs-Tools** und klicken Wörterbuch an.

Siehe auch

Anwendungen lokalisieren (siehe Seite 1437)

2.408 Code-Explorer

Ansicht ▶ Code-Explorer

Verwenden Sie den Code-Explorer zum Navigieren durch Ihre Unit-Dateien. Das Fenster des Code-Explorer enthält ein Baumdiagramm, das alle Typen, Klassen, Eigenschaften, Methoden, globalen Variablen und globalen Routinen enthält, die in Ihrer Unit definiert sind. Ferner sind hier die übrigen in der `uses`-Klausel aufgeführten Units aufgeführt. Klicken Sie mit der rechten Maustaste einen Eintrag im Code-Explorer an, um dessen Kontextmenü einzublenden.

Wenn Sie einen Eintrag im Code-Explorer markieren, wird der Cursor im Quelltext-Editor an die Stelle gesetzt, an der der Eintrag implementiert ist. Wenn Sie umgekehrt den Cursor im Quelltext-Editor bewegen, wird jeweils der entsprechende Eintrag im Code-Explorer markiert.

Der Code-Explorer verwendet folgende Symbole:

Symbol	Beschreibung
	Klassen
	Interfaces
	Units
	Konstanten oder Variablen (einschließlich Felder)
	Methoden oder Routinen: Prozeduren (grün)
	Methoden oder Routinen: Funktionen (gelb)
	Eigenschaften
	Typen

Tip: Zum Ändern der Einstellungen des Code-Explorers wählen Sie **Tools ▶ Optionen ▶ Delphi-Optionen ▶ Explorer**.

2.409 Symbolleisten anpassen

Ansicht ▶ Symbolleisten ▶ Anpassen

Verwenden Sie dieses Dialogfeld zum Ändern der Symbolleistenkonfiguration. In diesem Dialogfeld können Sie Schaltflächen für Symbolleisten hinzufügen, entfernen und neu anordnen.

Seite Symbolleisten

Die Seite **Symbolleisten** listet die Symbolleisten auf, die Sie anzeigen, ausblenden oder zurücksetzen können.

Element	Beschreibung
Symbolleisten	Führt die verfügbaren Symbolleisten, wie. z.B. Standard, Debug und Desktop auf.
Zurücksetzen	Setzt alle markierten Symbolleisten auf ihre Standardkonfiguration zurück.

Seite Anweisungen

Die Seite **Anweisungen** zeigt die Menübefehle an, die Sie per Drag&Drop auf eine Symbolleiste ziehen können.

Element	Beschreibung
Kategorien	Führt die verfügbaren Menüs, wie. z.B. Debug und Start auf.
Anweisungen	Listet alle Befehle für die im Listenfeld Kategorien markierten Menüs auf. Das Symbol links vom Menübefehl zeigt, welche Schaltfläche für diesen Befehl in die Symbolleiste eingefügt wird.

Seite Optionen

Auf der Seite **Optionen** können Sie festlegen, ob in der IDE Kurzhinweise und Tastenkürzel für Symbolleistenschaltflächen angezeigt werden.

Element	Beschreibung
Kurzhinweise anzeigen	Zeigt Kurzhinweise für Symbolleistenschaltflächen an, wenn Sie den Mauszeiger über die Schaltfläche bewegen.
Tastenkürzel Kurzhinweisen in	Zeigt in den Kurzhinweisen zu Symbolleistenschaltflächen Tastenkürzel an.

Siehe auch

Symbolleisten anpassen (☞ siehe Seite 63)

2.410 Daten-Explorer

Ansicht ▾ Daten-Explorer

Verwenden Sie dieses Dialogfeld, um einen neuen Verbindung hinzuzufügen oder vorhandene Verbindungen zu ändern, zu löschen oder umzubenennen. Sie können server-spezifische Schemaobjekte von Datenbanken, wie Tabellen, Felder, Stored Procedure-Definitionen, Trigger und Indizes durchsuchen. Weiterhin können Sie Daten aus einer Datenquelle in jedes beliebige Projekt ziehen und so schnell Ihre Datenbankanwendung erstellen. Welche Befehle im Daten-Explorer zur Verfügung stehen, hängt von dem in der Hierarchie markierten Objekt ab. Für die folgenden Knoten stehen Befehl zur Verfügung:

- Provider-Typen
- Provider-Verbindungen
- Tabellenknoten
- Einzelne Tabellen
- Einzelne Sichten
- Einzelne Stored Procedures

Befehle für Provider-Typen

Die folgenden Befehle sind verfügbar, wenn Sie einen Knoten für Provider-Typen, wie z.B. DB2 und Interbase, mit der rechten Maustaste anklicken:

Element	Beschreibung
Aktualisieren	Initialisiert alle für den ausgewählten Provider definierten Verbindungen erneut.
Neue Verbindung hinzufügen	Fügt dem Daten-Explorer einen neuen Verbindung hinzu.
Datenmigration	Öffnet eine Registerseite des Daten-Explorers für die Datenmigration im Quelltext-Editor. Auf dieser Seite können Sie eine oder mehrere Tabellen in einer Provider-Quellverbindung auswählen und eine Zielverbindung bestimmen, zu der die Tabellen migriert werden sollen. Klicken Sie auf Ausführen, um die Tabellen zu übernehmen.

Befehle für einzelne Provider

Die folgenden Befehle sind verfügbar, wenn Sie einen Knoten für einzelne Provider-Verbindungen auswählen:

Element	Beschreibung
Aktualisieren	Initialisiert alle für den ausgewählten Provider definierten Verbindungen erneut.
Verbindung löschen	Löscht die aktuelle Verbindung.
Verbindung bearbeiten	Ändert die entsprechenden Werte im Editor.
Verbindung schließen	Schließt die aktuelle Verbindung.
Verbindung umbenennen	Benennt eine benannte Verbindung um.
SQL-Fenster	Öffnet eine Registerseite des Daten-Explorers für das Schreiben und Bearbeiten von SQL-Anweisungen im Quelltext-Editor. Auf dieser Seite können Sie SQL-Anweisungen schreiben, bearbeiten und ausführen. Wenn Sie das SQL ausführen, werden die Ergebnisse im unteren Teil der Seite angezeigt.

Befehle für Tabellenknoten

Die folgenden Befehle sind verfügbar, wenn Sie den Tabellenknoten für eine Verbindungen auswählen:

Element	Beschreibung
Aktualisieren	Initialisiert alle für den ausgewählten Provider definierten Verbindungen erneut.
Neue Tabelle	Öffnet eine Registerseite des Daten-Explorers für den Entwurf von Tabellen im Quelltext-Editor. Auf dieser Seite können Sie die Datenstruktur für eine neue Tabelle festlegen. Hier können Sie Spalten hinzufügen und entfernen und Spalteninformationen ändern. Sie können die folgenden Spalteninformationen ändern: Spaltenname, Datentyp, Genauigkeit, Größe und Nullable (ob die Spalte den Wert Null enthalten darf). Klicken Sie die Seite mit der rechten Maustaste an und wählen Sie Tabelle speichern, um Ihrer Datenbank die neue Tabelle hinzuzufügen.

Befehle für einzelne Tabellen

Die folgenden Befehle sind verfügbar, wenn Sie einzelne Tabellen auswählen:

Element	Beschreibung
Aktualisieren	Initialisiert alle für den ausgewählten Provider definierten Verbindungen erneut.
Daten aus Tabelle abrufen	Öffnet eine Registerseite des Daten-Explorers im Quelltext-Editor, die die Daten aus der markierten Tabelle anzeigt. Auf dieser Seite können Sie die Daten sortieren und modifizieren, die Änderungen werden aber nicht in die Datenbank zurückgeschrieben.
Tabelle verwerfen	Entfernt die markierte Tabelle samt aller Daten aus der Datenbank.
Tabelle ändern	Öffnet eine Registerseite des Daten-Explorers für den Entwurf von Tabellen im Quelltext-Editor. Auf dieser Seite können Sie die Datenstruktur für eine vorhandene Tabelle ändern. Hier können Sie Spalten hinzufügen und entfernen und Spalteninformationen ändern. Sie können die folgenden Spalteninformationen ändern: Spaltenname, Datentyp, Genauigkeit, Größe und Nullable (ob die Spalte den Wert Null enthalten darf). Wenn Sie diese Seite schließen, werden Sie gefragt, ob die Änderungen gespeichert werden sollen.
Tabelle kopieren	Kopiert die Tabellenstruktur und die Daten der markierten Tabelle.
Tabelle einfügen	Fügt die Tabellenstruktur und die Daten, die von einem Provider kopiert wurden, in den markierten Provider ein. Obwohl Sie eine Tabelle im Ziel-Provider markieren müssen, werden keine Daten überschrieben.

Befehle für Sichten

Die folgenden Befehle sind verfügbar, wenn Sie einzelne Sichten auswählen:

Element	Beschreibung
Aktualisieren	Initialisiert alle für den ausgewählten Provider definierten Verbindungen erneut.
Daten aus Sicht abrufen	Öffnet eine Registerseite des Daten-Explorers im Quelltext-Editor, die die Daten aus der markierten Sicht anzeigt. Auf dieser Seite können Sie die Daten sortieren und modifizieren, die Änderungen werden aber nicht in die Datenbank zurückgeschrieben.

Befehle für Stored Procedures

Die folgenden Befehle sind verfügbar, wenn Sie einzelne Stored Procedures auswählen:

Element	Beschreibung
Aktualisieren	Initialisiert alle für den ausgewählten Provider definierten Verbindungen erneut.
Parameter anzeigen	Öffnet eine Registerseite des Daten-Explorers für das Anzeigen und Bearbeiten Parametern für Stored Procedures im Quelltext-Editor. Die Stored Procedure kann von dieser Seite aus auch ausgeführt werden.

Siehe auch

[Borland-Datenprovider für Microsoft .NET](#)

[Daten zwischen Datenbanken migrieren](#)

[ISQLSchemaCreate](#)

[BdpCopyTable](#)

2.411 Modul hinzufügen

Verwenden Sie dieses Dialogfeld, um dem Fenster Module ein Modul hinzuzufügen. Module werden automatisch zum Fenster Module hinzugefügt, wenn sie in den Speicher geladen werden; wenn Sie aber die Ausführung zum Debuggen anhalten möchten, sobald das Modul erstmals in den Speicher geladen wird, müssen Sie es zunächst dem Module hinzufügen.

Geben Sie den Modulnamen (normalerweise eine DLL oder ein Package) in das Eingabefeld ein, oder klicken Sie auf Durchsuchen, und wählen Sie die gewünschte Datei aus.

2.412 Gruppe überwachter Ausdrücke hinzufügen

Verwenden Sie dieses Dialogfeld, um einen Namen für eine neue Gruppe überwachter Ausdrücke zu vergeben oder um einen Namen aus der Liste der bisher eingegebenen Gruppennamen auszuwählen.

Die Gruppe überwachter Ausdrücke wird der Liste überwachter Ausdrücke als Registerseite hinzugefügt.

2.413 Liste der Haltepunkte

Ansicht ▶ Debug-Fenster ▶ Haltepunkte

Verwenden Sie das Fenster Liste der Haltepunkte zum Anzeigen, Aktivieren oder Deaktivieren von Haltepunkten, die im aktuell geladenen Projekt gesetzt sind und zum Ändern der Bedingung, des Durchlaufzählers oder der dem Haltepunkt zugeordneten Gruppe. Wenn kein Projekt geladen ist, werden alle im aktiven Quelltext-Editor oder im Fenster CPU gesetzten Haltepunkte angezeigt.

Tip: Einige der im Kontextmenü der Liste der Haltepunkte enthaltenen Befehle stehen auch in der Symbolleiste Liste der Haltepunkte zur Verfügung.

Spalte	Beschreibung
<input checked="" type="checkbox"/>	Gibt an, ob der Haltepunkt aktiviert oder deaktiviert ist. Markieren Sie das Feld, um den Haltepunkt zu aktivieren. Zum Deaktivieren des Haltepunkts entfernen Sie die Markierung.
Dateiname/Adresse	Die Quelldatei für den Quelltexthaltepunkt oder die Adresse für den Adresshaltepunkt.
Zeile/Länge	Die Zeile im Quelltext für den Haltepunkt oder die Länge (Anzahl der zu überwachenden Byte) für den Datenhaltepunkt.
Bedingung	Gibt den Bedingungsausdruck an, der bei jedem Durchlauf ausgewertet wird. Klicken Sie auf einen Wert, um diesen zu bearbeiten.
Aktion	Gibt die dem Haltepunkt zugewiesene Aktion an.
Anzahl Durchgänge	Gibt den aktuellen Durchlauf und die Gesamtanzahl der für den Haltepunkt festgelegten Durchläufe an. Klicken Sie auf einen Wert, um diesen zu bearbeiten.
Gruppe	Gibt den Namen der Gruppe an, der der Haltepunkt zugeordnet ist. Klicken Sie auf einen Wert, um diesen zu bearbeiten.

Mit den folgenden Symbolen werden Haltepunkte im Fenster Liste der Haltepunkte dargestellt.

Symbol	Beschreibung
	Der Haltepunkt ist gültig und aktiviert.
	Der Haltepunkt ist gültig und deaktiviert.
	Der Haltepunkt wurde an einer ungültigen Position gesetzt, zum Beispiel in einem Kommentar, einer Leerzeile oder einer ungültigen Deklaration.

Kontextmenü, wenn kein Haltepunkt ausgewählt ist

Klicken Sie im Fenster Liste der Haltepunkte mit der rechten Maustaste (nicht auf einen Haltepunkt), um die folgenden Befehle anzuzeigen:

Element	Beschreibung
Hinzufügen	Öffnet Dialogfelder, in welchen Sie neue Haltepunkte festlegen können.
Alle löschen	Entfernt alle Haltepunkte. Dieser Befehl lässt sich nicht rückgängig machen.

Alle deaktivieren	Deaktiviert alle aktivierte Haltepunkte. Die definierten Einstellungen des Haltepunkts bleiben erhalten, das Programm wird jedoch nicht unterbrochen. Das Deaktivieren eines Haltepunktes ist hilfreich, wenn dieser zeitweise nicht benötigt wird, seine Definition aber für einen möglichen späteren Einsatz erhalten werden soll.
Alle aktivieren	Aktiviert alle deaktivierten Haltepunkte.
Gruppe deaktivieren	Deaktiviert die ausgewählte Haltepunktgruppe.
Gruppe aktivieren	Aktiviert die ausgewählte Haltepunktgruppe.
Andockbar	Legt fest, ob das Fenster Liste der Haltepunkte angedockt werden kann.
Immer im Vordergrund	Belässt das Fenster sichtbar, wenn es den Fokus verliert.

Kontextmenü, wenn ein Haltepunkt ausgewählt ist

Klicken Sie einen Haltepunkt mit der rechten Maustaste an, um die folgenden Befehle anzuzeigen.

Element	Beschreibung
Aktiviert	Aktiviert bzw. deaktiviert einen Haltepunkt.
Löschen	Entfernt einen Haltepunkt.
Quelltext anzeigen	Sucht bei Quelltexthaltepunkten einen Haltepunkt im Quelltext. Bei Adresshaltepunkten wird die Position im CPU-Fenster angezeigt.
Quelltext bearbeiten	Sucht bei Quelltexthaltepunkten einen Haltepunkt im Quelltext und öffnet den Quelltext-Editor. Bei Adresshaltepunkten wird die Position im CPU-Fenster angezeigt.
Eigenschaften	Zeigt das Dialogfeld Eigenschaften des Haltepunkts an, in dem Sie Haltepunkte bearbeiten können.
Haltepunkte	Zeigt ein Menü mit Haltepunktbefehlen an.
Immer im Vordergrund	Belässt das Fenster sichtbar, wenn es den Fokus verliert.
Andockbar	Aktiviert die Drag&Dock-Funktion für das Fenster Liste der Haltepunkte.

2.414 Aufruf-Stack

Ansicht ▶ Debug-Fenster ▶ Aufruf-Stack

Verwenden Sie das Fenster Aufruf-Stack, um die Funktionsaufrufe, die zur aktuellen Programmposition geführt haben, und die an die Funktionen übergebenen Argumente anzuzeigen. Das Fenster Aufruf-Stack listet die letzte aufgerufene Funktion gefolgt von den zuvor aufgerufenen Funktionen auf. Die erste aufgerufene Funktion befindet sich am Ende der Liste. Wenn für eine Funktion Debug-Informationen zur Verfügung stehen, werden nach der Funktion die beim Aufruf übergebenen Argumente aufgeführt.

Wenn Sie auf einen Eintrag im Fenster Aufruf-Stack doppelklicken, werden der Quelltext und die lokalen Einstellungen für den Frame angezeigt.

Zum Umschalten eines Haltepunktes für einen bestimmten Frame klicken Sie entweder auf das Haltepunktsymbol in der äußerst linken Spalte oder klicken den Frame mit der rechten Maustaste an und wählen aus dem Kontextmenü den Befehl Haltepunkt umschalten. Das Symbol in der äußerst linken Spalte des Fensters Aufruf-Stack zeigt Folgendes an:

- Ein blauer Pfeil (↗) steht für den obersten Stack-Frame.
- Ein rotes Häkchen (✓) zeigt an, dass für den Frame ein aktiver Haltepunkt gesetzt ist.
- Ein graues Häkchen (✗) zeigt an, dass für den Frame ein deaktiver Haltepunkt vorhanden ist.
- Ein blauer Kreis (●) gibt an, dass für den Frame Debug-Informationen (Symbole sind verfügbar) vorhanden sind.
- Ein grauer Kreis (○) gibt an, dass für den Frame keine Debug-Informationen (Symbole sind nicht verfügbar) vorhanden sind.

Element	Beschreibung
Quelltext anzeigen	Positioniert im Quelltext-Editor den Cursor auf den im Fenster Aufruf-Stack ausgewählten Funktionsaufruf, übergibt den Fokus aber nicht an den Quelltext-Editor.
Quelltext bearbeiten	Positioniert im Quelltext-Editor den Cursor auf den im Fenster Aufruf-Stack ausgewählten Funktionsaufruf und übergibt den Fokus an den Quelltext-Editor.
Lokale Variablen anzeigen	Zeigt im Fenster Lokale Variablen alle lokale Variablen an, die dem Funktionsaufruf zugeordnet sind, der aktuell im Fenster Aufruf-Stack ausgewählt ist.
Haltepunkt umschalten	Setzt einen deaktiven Haltepunkt oder deaktiviert einen gesetzten Haltepunkt. Haltepunktsymbole werden entsprechend des Symbolstatus des Frame in unterschiedlichen Farben dargestellt. Siehe die Beschreibung der Symbolfarben weiter oben.
Voll qualifizierte Namen anzeigen	Zeigt die vollständigen Pfade der Dateinamen an.
Immer im Vordergrund	Ermöglicht, dass das Fenster Aufruf-Stack immer sichtbar ist, auch wenn es nicht den Fokus hat.
Andockbar	Aktiviert die Drag&Dock-Funktion für das Fenster Aufruf-Stack.

Siehe auch

Compileren (siehe Seite 1351)

Quelltexthaltepunkte setzen und bearbeiten (siehe Seite 19)

2.415 CPU-Fenster

Ansicht ▶ Debug-Fenster ▶ CPU-Fenster

Verwenden Sie das Fenster CPU zum Anzeigen des Assemblierungssprachcodes für das aktuell im Debugger befindliche Programm. Dieses Fenster wird automatisch geöffnet, wenn die Programmausführung an einer Position anhält, für die kein Quellcode verfügbar ist.

Das Fenster CPU ist in die folgenden Bereiche unterteilt.

Bereich	Beschreibung
Adressstatus (im oberen Bereich des Fensters)	<p>Zeigt die effektive Adresse (falls verfügbar) und den an dieser Adresse gespeicherten Wert an. Wenn Sie beispielsweise eine Adresse auswählen, die einen Ausdruck in eckigen Klammern wie [eax+edi*4-0x0F] enthält, werden die referenzierte Stelle im Speicher und ihr aktueller Wert angezeigt.</p> <p>Die aktuelle Thread-ID wird ebenfalls angezeigt.</p>
Disassemblierungsausschnitt (obere linke Seite)	<p>Zeigt die Adresse, die hexadezimale Repräsentation der Maschinencodeanweisungen (Opcodes), und die Assemblierungsanweisungen für alle Quelltextzeilen an. Die Adresse ist der Offset für die Disassemblierungsmethode.</p> <p>Wenn Sie verwalteten Code debuggen, entsprechen die Assemblierungsanweisungen dem vom JIT-Compiler erzeugten nativen Code. Das vom Compiler erzeugte MSIL (Microsoft Intermediate Language) wird auch angezeigt. Beachten Sie bitte, dass es keine 1:1-Beziehung zwischen den nativen Codeanweisungen und den MSIL-Anweisungen gibt. MSIL-Anweisungen können nicht im Einzelschrittmodus getestet werden und es können auch keine Haltepunkte gesetzt werden.</p> <p>Falls Debug-Informationen verfügbar sind, so zeigt der Debugger den zu den Assemblierungsanweisungen gehörenden Quelltext an.</p> <p>Ein grüner Pfeil (↑) links neben der Adresse gibt die momentane Position des Ausführungspunkts an.</p> <p>Wenn die aktuelle Anweisung eine Transferanweisung ist (z.B. call oder jmp), zeigt ein Aufwärts- oder Abwärts-Pfeil die Zielrichtung der Transferanweisung an. Wenn sich das Ziel beispielsweise vor der aktuellen Anweisung befindet, wird ein Aufwärts-Pfeil angezeigt. Befindet sich das Ziel nach der aktuellen Anweisung, wird ein Abwärts-Pfeil angezeigt.</p> <p>Bei konditionalen Transferanweisungen (z.B. jz oder jle) wird nur ein Pfeil angezeigt, wenn die Bedingung true ist. Bei konditionalen Set-Anweisungen (z.B. seta oder setz) wird ein Links-Pfeil angezeigt, wenn die Bedingung true ist.</p>
Registerausschnitt (obere mittlere Seite)	<p>Zeigt die Inhalte der CPU-Register des 80386 und höherer Prozessoren an. Diese Register setzen sich aus acht 32 Bit-Vielzweckregistern und dem 32 Bit-Programmzähler (EIP) zusammen.</p> <p>Beim Debuggen von Win32-Code werden auch das Flags-Register (EFL) und die sechs Segmentregister angezeigt.</p> <p>Nach dem Ausführen einer Anweisung werden im CPU-Registerausschnitt etwaige Register, deren Wert sich gegenüber der letzten Programmpause geändert hat, rot markiert.</p>

Speicher auszugausschnitt (untere, linke Seite)	<p>Wird nur für unverwalteten Code angezeigt. Zeigt die Rohwerte an, die in adressierbaren Bereichen Ihres Programms enthalten sind. Dieser Ausschnitt zeigt die Speicheradressen, die aktuellen Werte im Speicher und eine ASCII-Präsentation der Werte im Speicher. Ganz links auf jeder Zeile wird die Anfangsadresse der Zeile angezeigt. Auf die Adresse folgt eine 8 Byte lange hexadezimale Liste der an dieser Speicheradresse befindlichen Werte. Jedes Byte im Speicher wird durch zwei Hexadezimalziffern wiedergegeben. Auf die hexadezimale Anzeige folgt eine ASCII-Anzeige des Speichers. Nicht druckbare Werte werden durch einen Punkt wiedergegeben.</p> <p>Betätigen Sie die STRG+Nach-Links-Taste oder die STRG+Nach-Rechts-Taste, um die Anfangsposition der Anzeige um ein Byte nach oben oder nach unten zu verschieben.</p>
Flags-Ausschnitt (obere rechte Seite)	<p>Wird nur für unverwalteten Code angezeigt. Zeigt den gegenwärtigen Zustand der im 32 Bit-Register EFL enthaltenen Flags und der Informations-Bits an. Nach Ausführung einer Anweisung markiert der Flags-Ausschnitt etwaige Flags, deren Wert sich gegenüber der letzten Programmpause geändert hat, rot.</p> <p>Der Prozessor verwendet die Bits in diesem Register, um bestimmte Operationen zu steuern und den Status des Prozessors nach Ausführung bestimmter Anweisungen anzuzeigen:</p> <p>Bewegen Sie die Maus über ein Flag, um den Flag-Namen anzuzeigen.</p>
Maschinen-Stack-Ausschnitt (untere, linke Seite)	<p>Wird nur für unverwalteten Code angezeigt. Zeigt die Rohwerte an, die in Ihrem Programm-Stack enthalten sind. Der Ausschnitt besteht aus drei Abschnitten: den Speicheradressen, den aktuellen Werten im Stack und einer ASCII-Darstellung der Werte im Stack. Ein grüner Pfeil gibt den Wert ganz oben im Aufruf-Stack an.</p>

Die einzelnen Bereiche des CPU-Fensters sind andockbar

Sie können jetzt über das Untermenü **Ansicht>Debug-Fenster>CPU-Fenster** einen einzelnen Bereich des Fensters CPU (wie z.B. Disassemblierung, Register oder Stack) öffnen. Die einzelnen Bereiche sind andockbare Ansichten, die Sie in der IDE verschieben können.

Das CPU-Fenster wird automatisch geschlossen

Wenn auf der Registerkarte **Tools>Optionen>Debugger-Optionen** die Option **Beim Debuggen implizit geöffnete Dateien automatisch schließen** aktiviert ist, wird das CPU-Fenster am Ende der Debug-Sitzung automatisch geschlossen. Wenn das CPU-Fenster allerdings das oberste Fenster ist, wird es nicht geschlossen.

Im Disassemblierungsausschnitt blättern

Verwenden Sie zum Blättern im Disassemblierungsausschnitt eine der folgenden Methoden:

- Betätigen Sie die **STRG+Nach-Links-Taste** und die **STRG+Nach-Rechts-Taste**, um die Anfangsposition der Anzeige um ein Byte nach oben oder nach unten zu verschieben. Beachten Sie, dass die Veränderung der Anfangsposition der Anzeige im Disassemblierungsausschnitt die Stelle verschiebt, an welcher der Debugger mit dem Disassemblieren des Maschinencodes beginnt.
- Klicken Sie ober- oder unterhalb des Bildlauffeldes auf der vertikalen Bildlaufleiste, um eine Bildschirmseite nach oben oder unten zu blättern. (Wegen der vielen Informationen, die im Disassemblierungsausschnitt zur Verfügung stehen, ist das Ziehen des Bildlauffeldes deaktiviert.)
- Verwenden Sie die Befehle des Kontextmenüs **Zu Adresse gehen**, **Zu aktuellem EIP**, **Verfolgen** und **Vorheriger** wie im folgenden Abschnitt beschrieben.

Kontextmenü

Die folgende Tabelle führt in alphabetischer Reihenfolge die Befehle für die Ausschnitte des Fensters CPU auf. Klicken Sie im CPU-Fenster mit der rechten Maustaste, um auf die folgenden Befehle zuzugreifen:

Element	Beschreibung
Haltepunkt-Eigenschaften	Zeigt das Dialogfeld Eigenschaften des Adresshaltepunkts an.

Aufrufende Routine	Positioniert den Disassemblierungs-Ausschnitt auf der Anweisung nach derjenigen, die den aktuellen Interrupt oder das aktuelle Unterprogramm aufgerufen hat. Hat die aktuelle Interrupt-Routine Datenelemente auf den Stack gelegt, so ist der Debugger möglicherweise nicht in der Lage, festzustellen, von wo aus die Routine aufgerufen wurde. Aufrufende Routine arbeitet am besten, wenn Sie die Option Stack-Frames unter Code-Erzeugung (auf der Seite Projekt ▸ Optionen ▸ Compiler) aktiviert ist.
Änderung	Ermöglicht Ihnen das Ändern der an der aktuellen Cursor-Position befindlichen Bytes und fordert Sie auf, ein Element des aktuell angezeigten Typs einzugeben.
Register wechseln	Zeigt das Dialogfeld Register wechseln an, in dem Sie einen neuen Wert für das Register eingeben können. Sie können die Ausdrucksauswertung in vollem Umfang nutzen, um neue Werte einzugeben. Denken Sie daran, dass Sie vor Hexadezimalwerten das Präfix \$ eingeben müssen.
Thread wechseln	Zeigt das Dialogfeld Thread auswählen an, in dem Sie aus der Thread-Liste einen neuen Thread auswählen können, der im Debugger getestet werden soll. Wenn Sie im Flags-Ausschnitt einen neuen Thread auswählen, geben alle Ausschnitte im CPU-Fenster den Zustand der CPU für diesen Thread wieder.
Kopieren	Die markierte Meldung wird in die Zwischenablage kopiert. Im Disassemblierungsfenster können Sie eine einzelne Anweisung auswählen oder die Taste UMSCHALT zur Auswahl mehrerer Anweisungen verwenden. In allen anderen Fenstern lässt sich nur jeweils ein Element zum Kopieren auswählen.
Register dekrementieren	Subtrahiert 1 vom Wert des momentan markierten Registers. Diese Option ermöglicht Ihnen das Testen von Fehlern der Sorte "knapp daneben", indem Sie geringfügige Änderungen an den Registerwerten vornehmen.
Anzeigen als	Formatiert die im Maschinen-Stack-Ausschnitt des CPU-Fensters angezeigten Daten. Wählen Sie aus den folgenden Formaten aus: Datentyp Anzeigeformat. Bytes Zeigt Daten als hexadezimale Bytes an Words Zeigt Daten als 2 Bytes lange hexadezimale Zahlen an DWords Zeigt Daten als 4 Bytes lange hexadezimale Zahlen an Singles Zeigt Daten als 4 Bytes lange Nachkommazahlen in wissenschaftlicher Notation an
Aktiviert	Ist nur beim Klicken mit der rechten Maustaste auf einen Haltepunkt verfügbar. Schaltet den Haltepunkt zwischen aktiviert und deaktiviert um.
Verfolgen	Positioniert den Ausschnitt an der Zieladresse der momentan markierten Ausweisung.
Zu Adresse gehen	Zeigt das Dialogfeld Position eingeben an, in dem Sie ein Symbol oder für verwalteten Code eine Adresse im JIT-Compiler-Format (Just In Time) eingeben können.
Zu aktuellem EIP	Positioniert das Fenster CPU am Ort des aktuellen Programmzählers (die vom Register EIP angegebene Adresse). Diese Stelle gibt die nächste von Ihrem Programm auszuführende Anweisung an.
Register inkrementieren	Addiert 1 zum Wert des momentan markierten Registers. Diese Option ermöglicht Ihnen das Testen von Fehlern der Sorte "knapp daneben", indem Sie geringfügige Änderungen an den Registerwerten vornehmen.
Gemischter IL-Code	Beim Debuggen von verwaltetem Code wird die Anzeige gewechselt, damit MSIL-Anweisungen angezeigt werden.
Gemischte Quelle	Schaltet das Anzeigeformat zwischen "Nur Assemblierungsanweisungen" und "Assemblierungsanweisungen und dem zugehörigen Quelltext" (wenn Debug-Informationen verfügbar sind) um.

Neuer EIP	<p>Verändert die Position des Anweisungszeigers (des Wertes des Registers EIP) auf die momentan im Disassemblierungs-Ausschnitt markierte Zeile. Verwenden Sie diesen Befehl, wenn Sie bestimmte Maschinenanweisungen übergehen möchten. Wenn Sie mit der Programmausführung fortfahren, beginnt die Ausführung an dieser Adresse.</p> <p>Dieser Befehl ist nicht dasselbe wie die Einzelschrittausführung von Anweisungen. Der Debugger führt keinerlei Anweisungen aus, die Sie möglicherweise übergehen.</p> <p>Verwenden Sie diesen Befehl mit äußerster Vorsicht. Leicht kann es passieren, dass Ihr System in einen instabilen Zustand gerät, wenn Sie Programmanweisungen übergehen.</p>
Weiter	Findet das nächste Vorkommen des Elements, nach dem Sie zuletzt im Speicherauszugs-Ausschnitt gesucht haben.
Zurück	Setzt das Fenster CPU auf den Zustand zurück, den es vor dem letzten Befehl Verfolgen hatte.
Zu aktueller Zeile gehen	Lässt Ihr Programm bei voller Geschwindigkeit zu der Anweisung laufen, die Sie im Fenster CPU markiert haben. Nachdem Ihr Programm angehalten wurde, können Sie diesen Befehl verwenden, um die Fehlersuche an einer bestimmten Programmanweisung wieder aufzunehmen.
Suchen	Zeigt das Dialogfeld Such-Bytes ein, mit dem Sie im Fenster CPU in Vorwärtsrichtung nach einem Ausdruck oder einer Liste von Bytes suchen können (Details erhalten Sie, wenn Sie im Dialogfeld Such-Bytes eingeben auf Hilfe klicken).
Adressen anzeigen	Schließt Anweisungsadressen ein.
Opcodes anzeigen	Schließt Anweisungs-Opcodes ein. Wählen Sie Auto, Immer oder Nie. Auto ist der Standardwert, der bewirkt, dass Opcodes immer angezeigt werden, wenn das Fenster breit genug zum Anzeigen der Opcode-Spalte ist.
Haltepunkt umschalten	Fügt an der aktuell markierten Adresse einen Haltepunkt hinzu oder entfernt ihn.
Flag umschalten	Die Flag- und Informations-Bits im Flags-Ausschnitt können jeweils einen der binären Werte 0 oder 1 enthalten. Dieser Befehl schaltet das markierte Flag zwischen diesen beiden Binärwerten um.
Anfang des Stack	Positioniert den Maschinen-Stack-Ausschnitt an der Adresse des Stack-Zeigers (der Adresse, die sich im Register ESP befindet).
FPU anzeigen	Steht nur beim Debuggen von Win32-Code zur Verfügung. Blendet das FPU-Fenster ein, in dem die Fließkommaregister, die MMX-Register und SSE-Register angezeigt werden.
Quelltext anzeigen	Aktiviert den Quelltext-Editor und positioniert den Einfügepunkt an derjenigen Quelltextzeile, die am genauesten der im Fenster CPU markierten disassemblierten Anweisung entspricht. Gibt es keinen zugehörigen Quelltext, so hat dieser Befehl keine Wirkung.
Register auf null	Setzt den Wert des aktuell markierten Registers auf 0.

2.416 Neuen Wert eingeben

Verwenden Sie dieses Dialogfeld dazu, den Wert zu bearbeiten, der sich an der aktuellen Cursorposition in den Ansichten CPU oder FPU befindet.

Dieses Dialogfeld erscheint, wenn Sie im Bereich Speicherauszug, Stack oder Register des Fensters CPU mit der rechten Maustaste klicken und aus dem Kontextmenü Ändern wählen bzw. dies im Bereich Register der Fensters FPU ausführen. Geben Sie hier den gewünschten Wert für das ausgewählte Element ein. Vor Hexadezimalwerten muss das Präfix \$ eingegeben werden.

Wenn Sie das Dialogfeld vom Speicherauszug- oder Stack-Bereich des CPU-Fensters aus öffnen, können Sie auch mehrere, durch Leerzeichen getrennte Werte eingeben. Beachten Sie, dass der Wert dem aktuellen, mit Anzeigen als angegebenen Anzeigeformat entsprechen muss.

Wenn Sie im Registerbereich des FPU-Fensters den Befehl Ändern wählen, sollten Sie einen 32-Bit-Hexadezimalwert eingeben (Dezimalzahlen sind zwar zulässig, aber nicht üblich).

2.417 Such-Bytes eingeben

Verwenden Sie dieses Dialogfeld, um im Disassemblierungsausschnitt des Fensters CPU einen Ausdruck oder eine Liste von Bytes in Vorwärtsrichtung zu suchen.

Element	Beschreibung
Eingabefeld	<p>Geben Sie eine Byte-Liste ein, um in dieser Reihenfolge nach mehreren Werten zu suchen. Stellen Sie hexadezimalen Werten <code>0x</code> voran. Wenn Sie beispielsweise <code>0x5D 0xC3</code> eingeben, geht der Debugger an die folgende Stelle:</p> <p><code>00000001 5D</code> <code>00000002 C3</code></p> <p>Alternativ dazu können Sie auch ein Dollarzeichen (<code>\$</code>) anstelle von <code>0x</code> verwenden.</p> <p>Sie können auch nach DWords suchen, müssen dann aber die Bytes vertauschen. Wenn Sie beispielsweise <code>0x1234</code>, eingeben, positioniert der Debugger den Ausschnitt an der Speicherposition <code>34 12</code>.</p>

2.418 Fenster Ereignisprotokoll

Ansicht>Debug-Fenster>Ereignisprotokoll

Verwenden Sie das Fenster Ereignisprotokoll zum Anzeigen von Haltepunkt-, Prozesssteuerungs-, Thread-, Modul- und Ausgabemeldungen, die während einer Debug-Sitzung auftreten.

Klicken Sie im Fenster Ereignisprotokoll mit der rechten Maustaste, um die folgenden Befehle anzuzeigen.

Element	Beschreibung
Ereignisse löschen	Entfernt alle Meldungen aus dem Fenster Ereignisprotokoll.
Ereignisse in Datei speichern	Zeigt das Dialogfeld Ereignisprotokoll als Datei speichern an, in dem Sie Meldungen in dem Fenster Ereignisprotokoll in eine Textdatei speichern können.
Kommentar hinzufügen	Zeigt das Dialogfeld Kommentar hinzufügen an, in dem Sie einen Kommentar an das Ende des Ereignisprotokolls anfügen können.
Eigenschaften	Zeigt die Seite Eigenschaften des Debugger-Ereignisprotokolls an, mit der Sie den Inhalt und das Erscheinungsbild des Fensters Ereignisprotokoll steuern können. Sie können diese Seite auch mit dem Befehl Tools>Optionen>Debugger>Optionen>Ereignisprotokoll anzeigen.
Immer im Vordergrund	Belässt das Fenster sichtbar, wenn es den Fokus verliert.
Andockbar	Aktiviert die Drag&Dock-Funktion für das Fenster Ereignisprotokoll.

2.419 Kommentar zu Ereignisprotokoll hinzufügen

Verwenden Sie diese Dialogfeld, um am Ende des Fensters Ereignisprotokoll einen Kommentar hinzuzufügen.

Element	Beschreibung
Kommentar	Geben Sie den Kommentar ein, der im Fenster Ereignisprotokoll erscheinen soll.

2.420 FPU

Ansicht ▶ Debug-Fenster ▶ FPU

Verwenden Sie das Fenster FPU, um den Inhalt der Fließkommaeinheit (Floating-point Unit) und der SSE-Register in der CPU anzuzeigen.

Element	Beschreibung
Anweisungszeiger (IPTR)	Zeigt die Adresse des Anweisungszeigers (IPTR), den Opcode und die Adresse des Operanden (OPTR) der zuletzt ausgeführten Fließkommaanweisung an.
FPU-Registerausschnitt	<p>Zeigt den Inhalt der Fließkommaregister (ST0 bis ST7) in aufsteigender Reihenfolge an. Nach diesen Informationen wird der Inhalt des Kontroll-, Status- und Tag-Worts angezeigt. Die Liste enthält für jedes Register folgende Informationen: Registername, Registerstatus und Registerwert.</p> <p>Der Registerstatus kann einer der folgenden Werte sein:</p> <p>Leer Gibt an, dass das Register ungültige Daten enthält. Wenn ein Register leer ist, wird kein Registerwert angezeigt, da sein Inhalt als ungültig betrachtet wird.</p> <p>Gültig Gibt an, dass das Register gültige Daten ungleich Null enthält.</p> <p>Spez. (Speziell) Gibt an, dass das Register gültige Daten enthält, die aber eine spezielle Bedingung repräsentieren, entweder NAN (Not A Number = keine Zahl), unendlich oder ein denormalisierter Wert sind.</p> <p>Der Status der einzelnen Register wird anhand des Tag-Worts und des elften bis dreizehnten Bits im Statuswort ermittelt. Wenn der Status eines Registers nicht Leer ist, wird sein Wert im Extended-Format (Long Double) direkt nach seinem Status angezeigt. Die Anzeige kann in verschiedenen Formaten erfolgen.</p> <p>Der Wert des Kontroll-, Status- und Tag-Worts wird nur hexadezimal angezeigt. Bei diesen Wörtern werden die in der letzten Ausführungsoperation geänderten Werte rot angezeigt.</p>
Kontroll-Flags-Auschnitt	<p>Enthält Flags des Kontrollworts der FPU. Die in der letzten Ausführungsoperation geänderten Flags werden rot angezeigt. Folgende Kontroll-Flags und Bit-Nummern im Kontrollwort sind möglich:</p> <p>IM Ungültige Operation (Exception), 0</p> <p>DM Denormalisierte Operation (Exception), 1</p> <p>ZM Division durch 0 (Exception), 2</p> <p>OM Überlauf (Exception), 3</p> <p>UM Unterlauf (Exception), 4</p> <p>PM Genauigkeit (Exception), 5</p> <p>PC Genauigkeitskontrolle, 8, 9</p> <p>RC Rundungskontrolle, 10, 11</p> <p>IC Endlichkeitskontrolle (veraltet), 12</p> <p>Wählen Sie das gewünschte Flag aus, und klicken Sie mit rechten Maustaste, um seinen Wert zu ändern. Bei Ein-Bit-Flags wird der Wert von 0 in 1 oder von 1 in 0 geändert. Bei den Flags, die aus mehreren Bits bestehen, werden die Werte der Reihe nach zugewiesen.</p>

Status-Flags-Ausschnitt	<p>Enthält die Flags des Statusworts der FPU. Die in der letzten Ausführungsoperation geänderten Flags werden rot angezeigt. Folgende Flags und Bit-Nummern im Kontrollwort sind möglich:</p> <p>IE Ungültige Operation (Exception), 0 DE Denormalisierte Operation (Exception), 1 ZE Division durch 0 (Exception), 2 OE Überlauf (Exception), 3 UE Unterlauf (Exception), 4 PE Genauigkeit (Exception), 5 SF Stack-Fehler, 6 ES Status der Fehlerzusammenfassung, 7 C0 Bedingungs-Code 0 (CF), 8 C1 Bedingungs-Code 1, 9 C2 Bedingungs-Code 2 (PF), 10 ST Anfang des Stack, 11-13 C3 Bedingungs-Code 3 (ZF), 14 BF FPU ausgelastet, 15</p> <p>Wählen Sie das gewünschte Flag aus, und klicken Sie mit rechten Maustaste, um seinen Wert zu ändern. Bei Ein-Bit-Flags wird der Wert von 0 in 1 oder von 1 in 0 geändert. Bei den Flags, die aus mehreren Bits bestehen, werden die Werte der Reihe nach zugewiesen.</p>
SSE-Bereich	<p>Zeigt die SSE-Register (Streaming SIMD Extensions) an.</p> <p>Klicken Sie im SSE-Bereich mit der rechten Maustaste und wählen Sie Anzeigen als, um das Anzeigeformat des Registerinhalts zu ändern.</p>

Kontextmenü

Klicken Sie im FPU-Fenster mit der rechten Maustaste, um auf die folgenden Befehle zuzugreifen:

Element	Beschreibung
Null	Setzt den Wert des ausgewählten Registers auf 0. Bei einem der sieben FPU-Register werden auch die Tag-Bits im Tag-Wort auf 01 gesetzt, um anzugeben, dass das Register Null enthält.
Leer	Setzt die Tag-Bits im Tag-Wort auf 11, um anzugeben, dass das Register leer ist. Dieser Befehl kann nicht verwendet werden, wenn es sich bei dem ausgewählten Register um das Kontroll-, Status- oder Tag-Wort handelt.
Änderung	<p>Zeigt das Dialogfeld Neuen Wert eingeben an, in dem Sie einen neuen Wert für das ausgewählte Register eingeben können. Bei den sieben FPU-Registern werden auch die Tag-Bits im Tag-Wort auf 00 gesetzt, um anzugeben, dass das Register einen gültigen Wert enthält.</p> <p>Der Wert, den Sie in das Dialogfeld Neuen Wert eingeben eingeben, muss mit dem bei dem Befehl Anzeigen als festgelegten Format übereinstimmen. Wenn beispielsweise das aktuell angezeigte Format Extended ist, sollten Sie einen <i>Extended</i>-Wert in das Dialogfeld Ändern eingeben.</p>
Anzeigen als	<p>Bestimmt, wie die Werte in Registern angezeigt werden. Die Befehle in den Untermenüs sind von der Auswahl bei dem Befehl Anzeigen abhängig.</p> <p>Bei FPU-Registern können die Optionen Word und Extended (Long Double), bei MMX-Registern Byte, Word, DWord (Double-Word) und QWord (Quad-Word) verwendet werden.</p> <p>Bei SSE-Registern sind die folgenden Werte möglich: Bytes, Words, DWords (Double-Words), QWords (Quad-Word), DQWords (Double-Quad-Words), Singles und Doubles.</p>
Radix	Nur verfügbar, wenn MMX-Register angezeigt werden. Bestimmt, wie die Werte in den MMX-Registern angezeigt werden. Die möglichen Werte sind Binär, Vorzeichenloser Dezimalwert, Hexadezimal und Dezimal mit Vorzeichen.

Anzeigen	<p>Schaltet die Anzeige im FPU-Bereich zwischen FPU- und MMX-Registern um.</p> <p>Fließkomma-Register zeigen die 10-Byte FPU-Register ST(0) bis ST(7) an. Sie können entweder als Extended-Werte (Long Double) oder als 5 DWord-Werte angezeigt werden.</p> <p>MMX-Register zeigen die 8-Byte MMX-Register MM0 bis MM7 an. Die Register können als 8-Byte-, 4-Wort-, 2-DWort oder 1-QWort-Wert im binären, dezimalen oder hexadezimalen Format (siehe Radix) angezeigt werden. MMX-Register können nur auf einem Rechner mit einem entsprechenden Prozessor verwendet werden.</p>
Flag umschalten	Ändert in den Bereichen Status-Flags und Kontroll-Flags den Wert des markierten Flags. Bei Ein-Bit-Flags wird der Wert von 0 in 1 oder von 1 in 0 geändert. Bei den Flags, die aus mehreren Bits bestehen, werden die Werte der Reihe nach zugewiesen.
Immer im Vordergrund	Zeigt das FPU-Fenster über allen anderen Fenster auf dem Desktop an.

2.421 Lokale Variablen

Ansicht > Debug-Fenster > Lokale Variablen

Verwenden Sie das Fenster Lokale Variablen, um im Debug-Modus die lokalen Variablen der aktuellen Funktion anzuzeigen. Zum Anzeigen von lokalen Variablen aus einem nicht aktuellen Frame wählen Sie einen Frame aus der Dropdown-Liste aus.

Klicken Sie im Fenster Lokale Variablen mit der rechten Maustaste, um die folgenden Befehle anzuzeigen.

Element	Beschreibung
Untersuchen	Zeigt Informationen über die aktuell ausgewählte Variable im Inspektor-Fenster an.
Immer im Vordergrund	Ermöglicht, dass das Fenster Lokale Variablen immer sichtbar ist, auch wenn es nicht den Fokus hat.
Andockbar	Aktiviert die Drag&Dock-Funktion für das Fenster Lokale Variablen.

Tip: Sie können dieses Fenster auch mit der Tastenkombination STRG+ALT+L anzeigen, wenn ein beliebiges IDE-Fenster den Fokus hat, sogar, wenn der Debug-Modus nicht aktiv ist. Das Fenster ist dann aber so lange leer, bis der Debugger angehalten wird. Halten Sie dieses Fenster während einer Debugging-Sitzung offen, zu überwachen, wie Ihr Programm die Werte von Variablen bei der Ausführung des Programms aktualisiert.

2.422 Fenster Module

Ansicht>Debug-Fenster>Module

Verwenden Sie das Fenster Module, um eine Liste aller im Debugger befindlichen Prozesse sowie eine Liste der Module, die aktuell von den einzelnen Prozessen geladen wurden, anzuzeigen. Das Fenster ist in die folgenden Bereiche unterteilt.

Bereich	Beschreibung
Modulausschnitt (obere, linke Seite)	Zeigt die Prozesse und Module in der Reihenfolge an, in der sie geladen wurden. Für jeden Prozess können ein oder mehrere Module vorhanden sein, die vom Prozess geladen werden. Beendete Prozesse und Module werden aus der Liste entfernt. ☞ gibt den aktuellen Prozess an. Klicken Sie zum Sortieren der Anzeige eine Spaltenüberschrift an.
Quelltextausschnitt (untere, linke Seite)	Falls Debug-Informationen verfügbar sind, werden die zum Erstellen des Moduls, das aktuell im Ausschnitt Module markiert ist, verwendeten Quelldateien angezeigt.
Bereichs-Browser (rechte Seite, nur bei verwaltetem Code)	Zeigt eine hierarchische Ansicht der in der Anwendung verwendeten Namespaces, Klassen und Methoden an. 📁 repräsentiert einen Namespace. 🌐 repräsentiert eine Klasse. 🔖 repräsentiert eine Methode.
Ausschnitt Einsprungspunkt (rechte Seite, nur bei unverwaltetem Code)	Zeigt den Namen und die Adressen der Einsprungpunkte für das aktuelle im Ausschnitt Module markierte Modul an. Der Eintrittspunkt wird nur dann angezeigt, wenn der zugehörige Quelltext gefunden wird. Klicken Sie zum Sortieren der Anzeige eine Spaltenüberschrift an. Bei der Image-Basisadresse zur Laufzeit handelt es sich um den Speicher-Offset (in Hexadezimalwerten), von dem aus das Modul tatsächlich geladen wird. Im Unterschied dazu gibt es eine bevorzugte Image-Basisadresse, die Sie im Dialogfeld Projekt>Optionen angeben können.

Kontextmenüs

Klicken Sie im Ausschnitt Module mit der rechten Maustaste, um den folgenden Befehl für unverwalteten Code anzuzeigen.

Element	Beschreibung
Beim Laden anhalten	Hält die Ausführung der Anwendung an, wenn sie das gewählte Modul in den Speicher lädt. Diese Einstellung wird nur vom Borland Win32-Debugger verwendet.
Symboltabelle neu laden	Zeigt das Dialogfeld Symboltabelle neu laden an, mit dem Sie die Debug-Symboltabelle in das Fenster Module laden können.
Modul hinzufügen	Zeigt das Dialogfeld Modul hinzufügen an, in dem Sie der Liste ein Modul hinzufügen können. Mit diesem Befehl fügen Sie einen Haltepunkt für das Laden eines Moduls hinzu, das aktuell nicht geladen ist. Diese Einstellung wird nur vom Borland Win32-Debugger verwendet.

Klicken Sie im Ausschnitt Quelltext mit der rechten Maustaste, um den folgenden Befehl anzuzeigen.

Element	Beschreibung
Quelltext bearbeiten	Aktiviert den Quelltext-Editor und positioniert ihn auf das ausgewählte Modul.

Klicken Sie im Ausschnitt Bereichs-Browser mit der rechten Maustaste, um die folgenden Befehle anzuzeigen.

Element	Beschreibung
Klasse durchsuchen	Zeigt das Tool Borland Reflection an, mit dem Sie die aktuell ausgewählte Klasse untersuchen können.
Quelltext bearbeiten	Steht nur für Methoden zur Verfügung. Aktiviert den Quelltext-Editor und positioniert ihn auf der Methode. Wenn Sie ein Modul auswählen, das nicht mit dem JIT-Compiler compiliert wurde, erhalten Sie allerdings die Meldung Kein nativer Code verfügbar .

Klicken Sie im Ausschnitt Einsprungspunkt mit der rechten Maustaste, um den folgenden Befehl anzuzeigen.

Element	Beschreibung
Zu Eintrittspunkt springen	Zeigt im CPU-Fenster den ausgewählten Eintrittspunkt an, wenn für den Eintrittspunkt kein Quelltext vorhanden ist. Wenn der entsprechende Quelltext gefunden wird, wird dieser angezeigt. Das Programm muss angehalten werden, bevor zu einem Eintrittspunkt gesprungen werden kann.

2.423 Datei nicht gefunden

Verwenden Sie dieses Dialogfeld zum Suchen einer Datei, die der Debugger nicht finden kann.

Element	Beschreibung
Pfad der Datei	Zeigt den Namen der Quelldatei an, die der Debugger nicht finden kann. Klicken Sie auf Durchsuchen, und wählen Sie die betreffende Datei aus, oder geben Sie den vollständigen Pfadnamen ein.
Verzeichnis Debug-Quellpfad hinzufügen	zum Fügt den Dateipfad an das Ende des Debug-Quellpfades an (in Projekt>Optionen>Debugger).

2.424 Fenster Thread-Status

Ansicht ▶ Debug-Fenster ▶ Thread-Status

Verwenden Sie das Fenster Thread-Status, um den Status aller im Debugger befindlichen Prozesse und Threads anzuzeigen.

Element	Beschreibung
Thread-ID	Zeigt den Prozessnamen, die vom BS zugewiesene Thread-ID und, wenn der Thread benannt ist, dessen Namen an.
Status	Gibt den Zustand des Thread an, der entweder Ausführbar, Angehalten, Blockiert oder Keiner lauten kann. Bei einem Prozess wird angezeigt, wie dieser erzeugt wurde: Abgespalten, Angehängt oder Prozessübergreifend angehängt.
Status	Gibt den Thread-Status als eines der folgenden Elemente an: Haltepunkt - Der Thread wurde aufgrund eines Haltepunkts angehalten. Fehlgeschlagen - Der Thread wurde aufgrund einer Prozessor-Exception angehalten. Unbekannt - Der Thread ist nicht der aktuelle Thread, so dass sein Status unbekannt ist. Schrittweise ausgeführt - Der letzte Einzelschrittbefehl wurde erfolgreich abgeschlossen.
Speicherort	Gibt den Funktionsnamen oder die Adresse an, die dem Thread zugeordnet ist.

Tip: Der aktuelle Prozess wird durch einen grünen Pfeil gekennzeichnet. Nicht-aktuelle Prozesse sind an einem hellblauen Pfeil zu erkennen.

Der aktuelle Prozess und der aktuelle Thread bilden den Kontext für die nächste Benutzeraktion (Start, Pause, Zurücksetzen).

Kontextmenü

Klicken Sie im Fenster Thread-Status mit der rechten Maustaste, um die folgenden Befehle anzuzeigen.

Element	Beschreibung
Quelltext anzeigen	Zeigt im Quelltext-Editor die entsprechende Stelle im Quelltext der gewählten Thread-ID an, macht den Quelltext-Editor jedoch nicht zum aktiven Fenster.
Zu Quelltext gehen	Zeigt im Quelltext-Editor die entsprechende Stelle im Quelltext der gewählten Thread-ID an und macht den Quelltext-Editor zum aktiven Fenster.
Aktuell	Macht den gewählten Thread zum aktiven Thread, falls er es nicht bereits ist. Falls der Thread nicht bereits Teil des aktiven Prozesses ist, wird außerdem sein Prozess der aktive.
Prozess beenden	Beendet den Prozess, falls ein Prozess gewählt ist, oder den Prozess, zu dem der Thread gehört, falls ein Thread gewählt ist.
Prozess abtrennen	Beendet den Prozess, falls ein Prozess gewählt ist, oder den Prozess, zu dem der Thread gehört, falls ein Thread gewählt ist.
Prozess unterbrechen	Beendet den Prozess, falls ein Prozess gewählt ist, oder den Prozess, zu dem der Thread gehört, falls ein Thread gewählt ist. Diese Option ist nur verfügbar, wenn der Prozess ausgeführt wird.
Prozesseigenschaften	Öffnet ein Dialogfeld, in dem Sie Debugger-Optionen vorübergehend für einen bestimmten Prozess festlegen können.
Andockbar	Aktiviert die Drag&Dock-Funktion für das Fenster Thread-Status.

2.425 Fenster Liste überwachter Ausdrücke

Ansicht ▶ Debug-Fenster ▶ Überwachter Ausdrücke

Die Liste überwachter Ausdrücke zeigt den aktuellen Wert des überwachten Ausdrucks basierend auf dem Sichtbarkeitsbereich der Ausführungspunkt an. Die Liste überwachter Ausdrücke besteht aus mehreren Registerseiten, wobei jede Seite eine bestimmte Gruppe von Ausdrücken enthält. Während des Debuggens wird nur die Gruppe auf der aktiven Seite überwacht.

Tip: Um einen überwachten Ausdruck schnell zu aktivieren oder zu deaktivieren, markieren Sie das Kontrollkästchen neben dem Ausdruck.

Element	Beschreibung
Name des Ausdrucks	Zeigt den Ausdruck an, der überwacht wird.
Wert	Enthält den aktuellen Wert des eingegebenen Ausdrucks.

Anmerkung: Wenn sich der Ausführungspunkt an eine Stelle bewegt, an der eine der Variablen im Ausdruck undefiniert ist (außerhalb des Sichtbarkeitsbereiches), wird der gesamte überwachte Ausdruck undefiniert. Befindet sich der Ausführungspunkt wieder im Gültigkeitsbereich des Ausdrucks, wird im Fenster Liste überwachter Ausdrücke der aktuelle Wert des Ausdrucks angezeigt.

Tip: Durch das Gruppieren von Ausdrücken können Sie verhindern, dass bereichsüberschreitende Ausdrücke die Überwachung verlangsamen.

Kontextmenü

Klicken Sie im Fenster Liste überwachter Ausdrücke mit der rechten Maustaste, um die folgenden Befehle anzuzeigen.

Element	Beschreibung
Ausdruck bearbeiten	Öffnet das Dialogfeld Darstellung überwachter Ausdrücke, in dem Sie die Eigenschaften eines überwachten Ausdrucks ändern können.
Ausdruck hinzufügen	Öffnet das Dialogfeld Darstellung überwachter Ausdrücke, in dem Sie einen überwachten Ausdruck erstellen können.
Ausdruck einschalten	Aktiviert einen deaktivierten überwachten Ausdruck.
Ausdruck ausschalten	Deaktiviert einen überwachten Ausdruck. Deaktivierte Ausdrücke werden nicht geprüft, während Sie in Einzelschritten durch das Programm gehen oder es ausführen. Die Einstellungen deaktivierter Ausdrücke bleiben definiert. Das Deaktivieren überwachter Ausdrücke erhöht die Ausführungsgeschwindigkeit des Debuggers.
Ausdruck löschen	Entfernt einen überwachten Ausdruck. Dieser Befehl lässt sich nicht rückgängig machen.
Wert des Ausdrucks kopieren	Kopiert den Text aus der Spalte Wert des markierten Ausdrucks in die Zwischenablage.
Name des Ausdrucks kopieren	Kopiert den Text aus der Spalte Name des Ausdrucks des markierten Ausdrucks in die Zwischenablage.
Alle Ausdrücke einschalten	Aktiviert alle deaktivierten überwachten Ausdrücke.

Alle Ausdrücke ausschalten	Deaktiviert alle aktivierten überwachten Ausdrücke.
Alle Ausdrücke löschen	Entfernt alle überwachten Ausdrücke.
Gruppe hinzufügen	Zeigt ein Dialogfeld an, in dem Sie eine überwachte Gruppe benennen und sie der Liste der überwachten Ausdrücke als neues Register hinzufügen können.
Gruppe löschen	Löscht eine Gruppe zu überwachender Ausdrücke.
Ausdruck in Gruppe verschieben	Verschiebt einen oder mehrere ausgewählte Ausdrücke in eine andere Gruppe.
Immer im Vordergrund	Belässt das Fenster sichtbar, wenn es den Fokus verliert.
Spaltenüberschrift anzeigen	Blendet die Spaltenüberschriften Name des Ausdrucks und Wert ein oder aus.
Untersuchen	Zeigt Informationen über den derzeit markierten Ausdruck an.
Andockbar	Aktiviert die Drag&Dock-Funktion für das Fenster Liste überwachter Ausdrücke.

2.426 Gespeicherten Desktop löschen

Ansicht▸**Desktops**▸**Desktop löschen**

Verwenden Sie dieses Dialogfeld, um einen gespeicherten Desktop aus der Liste auszuwählen und ihn mit der Schaltfläche **Löschen** zu entfernen.

2.427 Symbolleiste Desktop

Verwenden Sie die Symbolleiste Desktop, um ein vorhandenes Desktop-Layout auszuwählen oder um die aktuellen Einstellungen als Desktop-Layout zu speichern.

Element	Beschreibung
Auswahlliste	Führt die ausgelieferten und benutzerdefinierten Desktop-Layouts auf. Klicken Sie das gewünschte Desktop-Layout an.
Aktuellen Desktop speichern	Zeigt das Dialogfeld Desktop speichern an, mit dem Sie die aktuellen Desktop-Einstellungen benennen und speichern können.
Debug-Desktop einstellen	Das aktuelle Layout wird als Debug-Desktop verwendet, der zur Laufzeit automatisch angezeigt wird.

Tip: Sie können auch [Ansicht>Desktops](#) wählen, um Ihre Desktop-Einstellungen zu verwalten.

Siehe auch

Desktop-Layouts speichern (siehe Seite 72)

2.428 Datei-Browser

Ansicht ▾ Datei-Browser

Verwenden Sie diesen andockbaren Datei-Browser im Windows Explorer-Stil zum Anzeigen von Dateien und Verzeichnissen oder zum Ausführen von einfachen Dateioperationen, während die IDE ausgeführt wird.. Der Datei-Browser unterstützt die Standardoptionen von Windows-Kontextmenüs sowie die folgenden RAD Studio-spezifischen Befehle:

Element	Beschreibung
Mit RAD Studio öffnen	Öffnet die ausgewählte Datei in der IDE.
Dem Projekt hinzufügen	Fügt dem aktuellen Projekt die ausgewählte Datei hinzu.

Siehe auch

Verwenden des Datei-Browsers (↗ siehe Seite 81)

2.429 Dem Wörterbuch hinzufügen

Verwenden Sie dieses Dialogfeld, um Strings in der ausgewählten Unit dem Übersetzungswörterbuch hinzuzufügen. Dieses Dialogfeld wird angezeigt, wenn Sie auf der Registerseite Arbeitsbereich des Translation-Managers einen Knoten mit der rechten Maustaste anklicken und den Befehl Strings dem Wörterbuch hinzufügen wählen.

Um einzelne Strings und nicht Strings für eine gesamte Unit hinzuzufügen, klicken Sie den String im Translation-Manager mit der rechten Maustaste an und wählen **Wörterbuch▶Strings dem Wörterbuch hinzufügen**.

Die folgenden Optionen legen die Kriterien für das Hinzufügen von Strings fest.

Element	Beschreibung
Status	Fügt Strings basierend des in der Spalte Status der Registerseite Arbeitsbereich angezeigten Status ein. Wählen Sie den hinzuzufügenden Status aus.
Aktion bei Doppelten	Legt fest, welche Aktion ausgeführt wird, wenn für einen String zwei Übersetzungen im Wörterbuch gefunden werden. Überspringen fügt den String nicht hinzu. Hinzufügen fügt den String dem Wörterbuch hinzu, wenn kein übersetzter String für den Original-String vorhanden ist. Immer hinzufügen fügt den String dem Wörterbuch immer hinzu, auch wenn der String bereits im Wörterbuch enthalten ist. Ersetzen überschreibt den vorhandenen String mit dem neuen String. Auswahl anzeigen zeigt ein Dialogfeld mit Auswahlmöglichkeiten an.
Mit Kontextinformationen	Fügt den Unit-Pfad und den in der Spalte Id der Registerseite Arbeitsbereich angezeigten Wert in das Übersetzungswörterbuch ein. Diese Kontextinformationen werden in der Statuszeile angezeigt, wenn ein String im Übersetzungswörterbuch markiert ist.
Wert	Legt fest, ob ein String basierend auf Änderungen des Originalwertes hinzugefügt wird. Geändert fügt den String nur dann ein, wenn der Original- und der übersetzte Wert unterschiedlich sind. Unverändert fügt den String auch dann ein, wenn der Original- und der übersetzte Wert gleich sind. Keine Überprüfung fügt den String ungeachtet dessen ein, ob er verändert wurde oder nicht, solange er den anderen in diesem Dialogfeld festgelegten Kriterien entspricht.
Kommentar	Fügt Strings basierend des in der Spalte Kommentar der Registerseite Arbeitsbereich angezeigten Textes ein oder schließt sie aus. Geben Sie den Kommentartext in das Eingabefeld ein und markieren Sie Einbeziehen, um Strings mit einem übereinstimmenden Kommentar hinzuzufügen, oder markieren Sie Nicht einbeziehen, um diese Strings zu übergehen.

Tip: Zum Setzen von allgemeinen Optionen für das Übersetzungswörterbuch wählen Sie **Tools▶Optionen▶Optionen für Übersetzungs-Tools** und klicken Wörterbuch an.

Siehe auch

Anwendungen lokalisieren (siehe Seite 1437)

2.430 Fenster: Meldungen

Diese Ansicht zeigt Meldungen wie Compiler-Fehler und -Warnungen an. Sie können eine oder mehrere Zeilen aus dem Fenster Meldungen in der Zwischenablage anzeigen.

Die Registerkarte Build zeigt den Build-Befehl an. Die Registerkarte Ausgabe zeigt die Build-Ausgabemeldungen an. Auf der Seite [Tools>Optionen>Umgebungsoptionen](#) können Sie Verbosity-Stufe für die Build-Ausgabe festlegen.

2.431 Objektinspektor

Ansicht▶Objektinspektor

Verwenden Sie den Objektinspektor zum Setzen der Eigenschaften und Ereignisse für das aktuell ausgewählte Objekt.

Register	Beschreibung
Eigenschaften	Zeigt die Eigenschaften des im Formular aktuell markierten Objekts an.
Ereignisse	Zeigt die Ereignisse des im Formular aktuell markierten Objekts an.

Kontextmenü

Klicken Sie im Objektinspektor mit der rechten Maustaste, um die folgenden Befehle anzuzeigen.

Element	Beschreibung
Ansicht	Filtert die Anzeige der Eigenschaften oder Ereignisse.
Arrange	Sortiert die Eigenschaften oder Ereignisse nach Namen oder Kategorien.
Geerbte Einstellungen wiederherstellen	Setzt die Eigenschaftseinstellung wieder zurück auf ihren ursprünglichen, geerbten Wert.
Einblenden	Blendet die Untereigenschaften (Untereignisse) der ausgewählten Eigenschaft (des Ereignisses) ein.
Ausblenden	Blendet die Untereigenschaften (Untereignisse) der ausgewählten Eigenschaft (des Ereignisses) aus.
Schließen	Schließt den Objektinspektor. Um ihn erneut anzuzeigen, wählen Sie Ansicht▶Objektinspektor
Hilfe	Zeigt dieses Hilfethema an.
Eigenschaften	Zeigt das Dialogfeld Eigenschaften des Objektinspektors an, in dem Sie das Erscheinungsbild des Objektinspektors ändern können.
Immer im Vordergrund	Zeigt den Objektinspektor über allen anderen Fenstern des Desktops an.
Andockbar	Aktiviert die Drag&Dock-Funktion für den Objektinspektor.

Siehe auch

Eigenschaften und Ereignisse festlegen (☞ siehe Seite 77)

2.432 Projektverwaltung

Ansicht ▶ Projektverwaltung

Verwenden Sie die Projektverwaltung, um den Inhalt der aktuellen Projektgruppe und aller enthaltenen Projekte anzuzeigen und zu organisieren. Sie können verschiedene Projektverwaltungsaufgaben, wie z.B. Dateien hinzufügen, entfernen und compilieren, ausführen.

Anmerkung: Einige der hier beschriebenen Funktionen sind nur in bestimmten Editionen des Produkts verfügbar. Beispielsweise stehen einige Funktionen in der Projektverwaltung nur für die C++ Personality zur Verfügung.

Element	Beschreibung
Projektliste	Zeigt die Projekte in der aktuellen Projektgruppe an.
Neu	Zeigt das Dialogfeld Objektgalerie an, mit dem Sie der aktuellen Projektgruppe ein neues Projekt hinzufügen können.
Entfernen	Entfernt das ausgewählte Projekt aus der aktuellen Projektgruppe.
Aktivieren	Zeigt das ausgewählte Projekt in der IDE vor anderen Projekten an, so dass Sie Änderungen vornehmen können. Sie können auf das Projekt auch doppelklicken, um es zu aktivieren. Das aktive Projekt wird in Fettschrift angezeigt.
Datei	Zeigt eine Baumhierarchie aller Dateien in dem Projekt oder der Projektgruppe an. Klicken Sie auf das Pluszeichen (+) bzw. das Minuszeichen (-), um alle Quelldateien des Projekts ein- bzw. auszublenden.

Gemeinsame Befehle des Kontextmenüs

Die Projektverwaltung hat je nachdem, was Sie auswählen (Datei, Projekt, Projektgruppe usw.), unterschiedliche Kontextmenüs. Die meisten Kontextmenüs enthalten jedoch die folgenden Menübefehle.

Element	Beschreibung
Automatisch ausblenden	Blendet die Baumstruktur des Projekts aus, nachdem eine Operation abgeschlossen wurde.
Andockbar	Dockt das Fenster der Projektverwaltung an andere Tool-Fenster, wie z.B. an den Quelltext-Editor an. Deaktivieren Sie diese Option, um die Projektverwaltung als frei platzierbares Fenster anzuzeigen.
Pfad anzeigen	Fügt der Projektverwaltung zum Anzeigen des Pfades von Dateien, Projekten und Projektgruppen das Feld Pfad hinzu.
Statuszeile	Zeigt den vollständigen Pfadnamen der markierten Datei am unteren Rand der Projektverwaltung an.
Immer im Vordergrund	Zeigt die Projektverwaltung über allen anderen Fenstern des Desktops an.
Symbolleiste	Blendet die Symbolleiste am oberen Rand der Projektverwaltung ein- oder aus.

Kontextmenü für Projektgruppen

Klicken Sie eine Projektgruppe mit der rechten Maustaste an, um die folgenden Befehle anzuzeigen.

Element	Beschreibung
Neues hinzufügen Projekt	Zeigt das Dialogfeld Objektgalerie an, mit dem Sie ein neues Projekt erstellen und es der aktuellen Projektgruppe hinzufügen können.
Existierendes hinzufügen Projekt	Öffnet das Dialogfeld Projekt öffnen an, in dem Sie der aktuellen Projektgruppe ein vorhandenes Projekt hinzufügen können.
Anpassen...	Öffnet das Dialogfeld Neues Menü anpassen, in dem Sie Einträge, einschließlich von Menütrennlinien, auswählen und dem Menü Neue hinzufügen in der Projektverwaltung hinzufügen können.
Projektgruppe speichern	Speichert die aktuelle Projektdatei (.bdsgroup) für die Projektgruppe. Verwenden Sie diesen Befehl nach dem Hinzufügen, dem Entfernen oder dem Ändern der Reihenfolge von Projekten in einer Projektgruppe.
Projektgruppe speichern unter	Zeigt das Dialogfeld Speichern unter an, mit dem Sie die Projektgruppe unter einem neuen Namen und an einer neuen Position speichern können.
Umbenennen	Benennt die Projektgruppe um.
Konfigurations-Manager	Öffnet das Dialogfeld Manager für Build-Konfigurationen. Mit diesem Dialogfeld wenden Sie die aktive Konfiguration auf ein Projekt oder mehrere Projekte an.

Kontextmenü für Projekte

Klicken Sie mit der rechten Maustaste eine Projektdatei an, um das Kontextmenü zu öffnen.

Element	Beschreibung
Hinzufügen	Zeigt das Dialogfeld Dem Projekt hinzufügen an, in dem Sie dem ausgewählten Projekt Dateien hinzufügen können.
Neue hinzufügen: Unit	Fügt dem ausgewählten Projekt eine Compilierungs-Unit hinzu. Die erste hinzugefügte Unit erhält den Namen Unit01.cpp . Für jede weitere hinzugefügte Unit wird die Zahl um eins erhöht. Zum Umbenennen einer Unit und all ihrer Komponenten klicken Sie den Unit-Namen in der Projektverwaltung mit der rechten Maustaste an und wählen Umbenennen.
Neue hinzufügen: Formular	Fügt dem ausgewählten Projekt ein neues Formular hinzu und zeigt es im Quelltext-Editor an.
Neue hinzufügen: Passwort-Dialogfeld	Fügt in C++Builder dem ausgewählten Projekt einen PassWord.cpp -Knoten mit der Datei PassWord.dfm hinzu und richtet eine Template zum Erstellen einer PassWord-Unit in Ihrem Projekt ein.
Neue hinzufügen: Weitere...	Zeigt das Dialogfeld Objektgalerie mit den verfügbaren Kategorien, wie z.B. C++Builder-Dateien, an. Klicken Sie im linken Fensterbereich auf eine Kategorie. Im rechten Fensterbereich werden daraufhin die Elemente dieser Kategorie angezeigt, die Sie Ihrem Projekt hinzufügen können.
Neue hinzufügen: Verzeichnisansicht	Öffnet das Dialogfeld Verzeichnisansicht oder Ordner, mit dem Sie ein Verzeichnis auswählen können, das der Projektverwaltung hinzugefügt werden soll. Der Baumstruktur wird ein gelber Ordnerknoten hinzugefügt.
Neue hinzufügen: Virtueller Ordner	Fügt dem ausgewählten Projekt einen abgedunkelten, virtuellen Ordner hinzu. Virtuelle Ordner werden standardmäßig zuletzt in der Baumstruktur angezeigt. Mit dem Kontextmenü können Sie den virtuellen Ordner verwalten.
Neue hinzufügen: Anpassen	Zeigt das Dialogfeld Neues Menü anpassen an, in dem Sie das Menü Datei > Neu anpassen können.
Referenz hinzufügen	Zeigt das Dialogfeld Referenz hinzufügen an, in dem Sie dem Projekt Assemblierungen, COM-Typbibliotheken und Projektreferenzen hinzufügen können.
Web-Referenz hinzufügen	Zeigt das Dialogfeld Referenz hinzufügen an, in dem Sie Ihrer Client-Anwendung Webreferenzen und den Zugriff auf Web-Services hinzufügen können.
Datei entfernen	Zeigt das Dialogfeld Aus dem Projekt entfernen an, in dem Sie Dateien aus dem ausgewählten Projekt entfernen können.

Speichern	Speichert die geänderten Dateien im geöffneten Projekt unter ihrem aktuellen Dateinamen. Falls Sie versuchen, ein Projekt zu speichern, zu dem eine ungespeicherte Quelltextdatei gehört, öffnet das Programm das Dialogfeld Speichern unter, in dem Sie den neuen Dateinamen angeben können.
Speichern unter...	Zeigt das Dialogfeld Speichern unter für alle Compilierungs-Units an, die gespeichert werden müssen, und ermöglicht die Eingabe eines neuen Namens für jede Unit.
Umbenennen	Markiert den Projektknoten und ermöglicht die Eingabe eines neuen Namens oder die Bearbeitung des aktuellen Namens des Projekts.
Projekt entfernen	Entfernt das aktive Projekt aus seiner Projektgruppe. Das Entfernen des Ziels aus der aktuellen Projektgruppe wirkt sich auf die Projektdatei der Projektgruppe (.bdsgroup) aus; es entfernt keine Dateien von der Festplatte. Entfernen Sie daher ein Projekt aus der Projektgruppe, bevor Sie die Datei von der Festplatte löschen, so dass die Projektdatei entsprechend aktualisiert werden kann.
Aktivieren	Macht das aktuelle Projekt zum aktiven.
Bereinigen	Entfernt erzeugte Dateien, wie z.B. Objektcodedateien, aus dem Projekt.
Compilieren	Compiliert alle Dateien im aktuellen Projekt, die sich seit der letzten Erstellung geändert haben und alle Dateien, die davon abhängig sind. Sie können dafür auch Projekt▶[Projektname] compilieren wählen.
Erzeugen	Erzeugt alle Datei in dem Projekt unabhängig davon, ob sie geändert wurden oder nicht. Alternativ können Sie auch Projekt▶[Projektname] erzeugen wählen.
Schließen	Schließt alle geöffneten Dateien des aktiven Projekts. Vor dem Schließen der Datei fordert Sie die IDE zum Speichern etwaiger Änderungen auf. Haben Sie das Projekt oder eine andere Datei zuvor nicht gespeichert, wird das Dialogfeld Speichern unter geöffnet, in dem Sie den neuen Dateinamen eingeben können.
Früher erstellen	Verlagert ein Projekt in der Projektliste einer Projektgruppe nach oben, wodurch die Reihenfolge, in der die Projekte compiliert werden, geändert wird.
Später erstellen	Verlagert ein Projekt in der Projektliste einer Projektgruppe nach unten.
Ab hier alle compilieren	Compiliert nur die ausgewählten Projekte und alle darunter aufgeführten Projekte, falls sie seit der letzten Erzeugung geändert wurden. Um alle geänderten Projekte der Projektgruppe zu compilieren, wählen Sie Projekt▶Alle Projekte compilieren .
Ab hier alle erzeugen	Erzeugt nur das ausgewählte Projekt und alle darunter aufgeführten Projekte, unabhängig davon, ob sie geändert wurden. Um alle geänderten Projekte der Projektgruppe zu erzeugen, wählen Sie Projekt▶Alle Projekte erzeugen .
Abhängigkeiten	Öffnet das Dialogfeld Projektabhängigkeiten, in dem Sie die Reihenfolge ändern können, in der die Projekte in einer Projektgruppe erzeugt werden sollen.
Optionen	Öffnet das Dialogfeld Projektoptionen, in dem Sie die Anwendungs- und Compiler-Optionen für das ausgewählte Projekt ändern können.

Kontextmenü für Dateien

Klicken Sie mit der rechten Maustaste eine Datei des Projekts an, um das Kontextmenü zu öffnen. Der Typ der ausgewählten Datei bestimmt, welche Befehle im Menü verfügbar sind.

Element	Beschreibung
Referenz hinzufügen	Zeigt das Dialogfeld Referenz hinzufügen an, in dem Sie dem Projekt Assemblierungen, COM-Typbibliotheken und Projektreferenzen hinzufügen können.
Web-Referenz hinzufügen	Zeigt das Dialogfeld Referenz hinzufügen an, in dem Sie Ihrer Client-Anwendung Webreferenzen und den Zugriff auf Web-Services hinzufügen können.
Erzeugen	Ist eine Kurzform zum Compilieren der ausgewählten Datei.
Lokal kopieren	Kopiert die Assemblierung in das lokale Ausgabeverzeichnis. Per Vorgabe ist Lokal kopieren für Assemblierungen markiert, die sich nicht im GAC (Global Assembly Cache) befinden.

Lokale bearbeiten	Optionen	Zeigt für C++ ein verkürztes Dialogfeld Projektoptionen an, das nur die Seite Pfade und Definitionen, neun Seiten mit C++Compiler-Optionen und die Seite Build-Ereignisse enthält.
Öffnen		Öffnet die ausgewählten Dateien im Quelltext-Editor.
Vorverarbeiten		Führt den C++-Präprozessor (cpp32) aus.
Aus dem Projekt entfernen		Entfernt die markierten Dateien aus dem Projekt. Sie werden zum Bestätigen des Speicherns von Änderungen aufgefordert.
Speichern		Speichert Änderungen der ausgewählten Dateien unter ihren derzeitigen Namen.
Speichern unter		Zeigt das Dialogfeld Speichern unter an, mit dem Sie die ausgewählten Dateien unter neuen Namen und an neuen Positionen speichern können.
Abhängigkeiten anzeigen		Anzeige
Umbenennen		Benennt die Datei und alle zugehörigen Sekundärdateien um, die als untergeordnete Knoten in der Projektverwaltung erscheinen.

Kontextmenü für Build-Konfigurationen (C++)

Klicken Sie entweder in der Projektverwaltung den Knoten Build-Konfigurationen oder den Namen einer bestimmten Konfiguration im Knoten mit der rechten Maustaste an, um das Kontextmenü anzuzeigen.

Element	Beschreibung
Neue hinzufügen	Fügt der Projektverwaltung unter der übergeordneten Konfiguration eine untergeordnete Konfiguration auf der Basis der ausgewählten Konfiguration hinzu.
Speichern unter	Zeigt das Dialogfeld Speichern unter an, in dem Sie die ausgewählte Konfiguration an einer bestimmten Position speichern und die gespeicherte Datei umbenennen können.
Umbenennen	Ermöglicht das Umbenennen der ausgewählten Build-Konfiguration.
Löschen	Zeigt das Dialogfeld Bestätigen an, in dem Sie das Löschen der ausgewählten Konfiguration bestätigen müssen.
Aktivieren	Aktiviert die ausgewählte Build-Konfiguration als aktuelle Konfiguration für das Projekt. Die aktive Build-Konfiguration wird in Fettschrift angezeigt.
Optionsgruppe anwenden	Zeigt das Dialogfeld Optionsgruppe anwenden an, in dem Sie eine .optset-Datei auswählen und für die ausgewählte Build-Konfiguration anwenden können. Sie können die vorhandenen Optionswerte überschreiben, ersetzen oder beibehalten.
Bearbeiten	Zeigt das Dialogfeld Projekt>Optionen mit den Werten an, die in der ausgewählten Build-Konfiguration festgelegt sind.

2.433 Desktop speichern

Ansicht▶Desktops▶Desktop speichern

Verwenden Sie dieses Dialogfeld, um die Anordnung des aktuellen IDE-Desktops als Desktop-Layout zu speichern.

Element	Beschreibung
Aktuellen Desktop speichern unter	Geben Sie für den Desktop einen neuen Namen ein oder wählen Sie einen Namen aus der Dropdown-Liste aus.

2.434 Debug-Desktop einstellen

Ansicht▸**Desktops**▸**Debug-Desktop einstellen**

Verwenden Sie dieses Dialogfeld, um festzulegen, welches Desktop-Layout beim Debuggen verwendet werden soll.

Element	Beschreibung
Debug-Desktop	Wählen Sie ein Desktop-Layout aus der Dropdown-Liste aus.

2.435 Strukturansicht

Ansicht>Struktur

Verwenden Sie die Strukturansicht, um die Hierarchie von im Quelltext-Editor angezeigten Quelltext oder HTML oder der im Designer eingeblendeten Komponenten darzustellen. Wenn Sie in der Strukturansicht für Quelltext oder HTML auf ein Element doppelklicken, wird im Quelltext-Editor die entsprechende Stelle bzw. Deklaration angezeigt. Wenn Komponenten angezeigt werden, können Sie auf eine Komponente doppelklicken, um sie im Formular auszuwählen.

Syntaxfehler in Ihrem Quelltext werden im Knoten Fehler der Strukturansicht angezeigt. Klicken Sie einen Fehler doppelt an, um im Quelltext-Editor zu dem entsprechenden Quelltext zu wechseln. (Nicht für die C++-Entwicklung verwendbar.)

Tip: Um den Inhalt und das Erscheinungsbild der Strukturansicht anzupassen, wählen Sie **Tools>Optionen>Umgebungsoptionen>Explorer** und ändern die Einstellungen nach Bedarf.

Kontextmenü

Klicken Sie in der Strukturansicht mit der rechten Maustaste, um die folgenden Befehle anzuzeigen. Die im Kontextmenü enthaltenen Befehle sind davon abhängig, ob in der Strukturansicht Quelltext oder Komponenten angezeigt werden.

Element	Beschreibung
Neu	Fügt einen neuen Knoten in die Strukturansicht ein.
Umbenennen	Ändert den Namen des in der Strukturansicht markierten Knotens.
Bearbeiten	Zeigt ein Untermenü mit Befehlen zum Ändern, Ausschneiden, Kopieren, Einfügen, Löschen und Markieren von Steuerelementen im Designer an.
Element	Zeigt ein Untermenü mit Befehlen zum Nach-vorne- oder Nach-hinten-Setzen des markierten Elements im Designer an.
Eigenschaften	Zeigt das Dialogfeld Explorer-Optionen an, in dem Sie den Inhalt und das Erscheinungsbild der Strukturansicht ändern können.
Immer im Vordergrund	Zeigt die Strukturansicht über allen anderen Fenstern des Desktops an.
Andockbar	Aktiviert die Drag&Dock-Funktion für die Strukturansicht.

Symbolleiste (C++)

Die Strukturansicht enthält eine Symbolleiste für die C++-Anwendungsentwicklung, mit der Sie steuern können, wie der Inhalt der Strukturansicht angezeigt wird. Sie enthält die folgenden Schaltflächen:

Alphabetisch sortieren	Sortiert den Inhalt der Strukturansicht alphabetisch.
Nach Typ gruppieren	Gruppiert die Elemente in der Strukturansicht in Ordnern nach dem Typ .
Nach Sichtbarkeit sortieren	Gruppiert Klassen-Member in Ordnern nach ihrer Sichtbarkeit: public , protected , private und published . Bei C++ ist 'Klassen' eine generische Gruppe, die Klassen , Strukturen , Varianten und Templates umfasst.
Typ anzeigen	Zeigt in der Strukturansicht den Typ rechts neben dem Member an.
Sichtbarkeit anzeigen	Schaltet die Anzeige der Strukturansicht zwischen verschiedenen Sichtbarkeitsebenen um: Nur public anzeigen, public und protected anzeigen, public , protected und private anzeigen und alles anzeigen.

2.436 To-Do-Liste

Ansicht ▾ To-Do-Liste

Verwenden Sie dieses Dialogfeld zum Erstellen und Verwalten von To-Do-Listen.

Element	Beschreibung
Aktionseintrag	Diese Spalte enthält ein Kontrollfeld, ein Symbol und die Aufgabe. Das Kontrollfeld gibt an, ob die Aufgabe fertig gestellt wurde. Ein Fenstersymbol zeigt an, dass der Eintrag in die To-Do-Liste eingegeben wurde. Ein Unit-Symbol zeigt an, dass es sich um einen Kommentar in dem Quelltext handelt. Der Text ist die zu erledigende Aufgabe. Abgedunkelter Text gibt an, dass der Eintrag aus einer Quelldatei stammt, die Bestandteil des aktuellen Projekts, aber nicht im Quelltext-Editor geöffnet ist. Fetter Text zeigt an, dass die Quelle im Quelltext-Editor geöffnet ist. Sie können eine Datei durch Doppelklicken auf den Eintrag in den Editor laden.
Priorität	Legt die Wichtigkeit der Aufgabe fest. Geben Sie hier einen Wert zwischen 1 (höchste Priorität) und 5 (geringste Priorität) an.
Modul	Bei Einträgen, die als Quelltextkommentar hinzugefügt wurden, wird der Pfad und das Modul, aus dem der Kommentar stammt, angezeigt.
Eigentümer	Gibt die Person an, die für die Aufgabe verantwortlich ist.
Kategorie	Gibt den Typ der Aufgabe an, z.B. Benutzeroberfläche oder Fehlerbehandlung.

Tip: Klicken Sie zum Sortieren der Liste die Spaltenüberschrift an. Weitere Verarbeitungsoptionen stehen im Kontextmenü zur Verfügung.

Siehe auch

To-Do-Listen verwenden (↗ siehe Seite 82)

2.437 To-Do-Eintrag hinzufügen oder bearbeiten

Verwenden Sie dieses Dialogfeld zum Hinzufügen oder Ändern von To-Do-Listeneinträgen.

Element	Beschreibung
Text	Legt den Text des To-Do-Listeneintrags fest.
Priorität	Legt die Priorität des Vorgangs (1 bis 5) fest. Sie können den Wert direkt oder mithilfe des Wipptellers eingeben.
Eigentümer	Gibt die Person an, die für die Aufgabe verantwortlich ist. Geben Sie den Namen ein, oder wählen Sie einen Wert mithilfe der Auswahlliste aus.
Kategorie	Gibt den Typ der Aufgabe an, z.B. Benutzeroberfläche oder UI oder Interface-Implementierung. Geben Sie die Kategorie ein, oder wählen Sie einen Wert aus der Liste aus.

Siehe auch

To-Do-Listen verwenden ( siehe Seite 82)

2.438 To-Do-Liste filtern

Verwenden Sie dieses Dialogfeld, um zu steuern, welche Einträge in der To-Do-Liste angezeigt werden.

Element	Beschreibung
Filter nach	Entfernen Sie die Markierung von einer Kategorie, einem Besitzer oder einem Eintragstyp, um diese Einträge in der To-Do-Liste auszublenden.
Alle anzeigen	Markiert alle Einträge in der Liste Filter nach.

Siehe auch

To-Do-Listen verwenden ( siehe Seite 82)

2.439 Tabelleneigenschaften

Verwenden Sie dieses Dialogfeld zum Festlegen des Erscheinungsbildes der To-Do-Liste, wenn Sie aus dem Kontextmenü des Dialogfelds To-Do-Liste den Befehl **Kopieren als HTML-Tabelle** wählen.

Registerseite Tabelle

Legt die Eigenschaften für die HTML-Tabelle zum Anzeigen der To-Do-Liste fest.

Element	Beschreibung
Titel	Legt einen Titel für die Tabelle fest.
Rahmenbreite	Legt die Stärke (in Pixel) des Rahmens um die Tabelle fest.
Breite (Prozent)	Legt die Breite der Tabelle fest. Der Wert ist relativ zur Größe des Fensters.
Zellzwischenraum	Legt den Abstand zwischen dem Rand der Tabelle und den äußersten Spalten fest. Dieser Wert wird auch für den Abstand zwischen den einzelnen Zellen verwendet.
Innenabstand	Legt den Abstand zwischen dem Zellrand und dem Zellinhalt fest.
Hintergrundfarbe	Gibt eine Hintergrundfarbe für die HTML-Tabellenzellen an.
Ausrichtung	Gibt die Position der Tabelle im HTML-Dokument (links, rechts oder zentriert) an.

Registerseite Spalten

Legt die Eigenschaften der Spalten in der To-Do-Liste fest.

Element	Beschreibung
Spalte	Gibt die Spalte an, deren Eigenschaften festgelegt werden sollen.
Ausrichtung	Legt die Textausrichtung in der Spalte (links, rechts oder zentriert) fest.
Vertikale Ausrichtung	Legt die Textausrichtung in der Zelle (oben, Mitte, unten) fest.
Titel	Gibt die Spaltenüberschrift an.
Breite	Die Breite der Spalte (in Prozent) relativ zur Tabellenbreite.
Höhe	Legt die Zellenhöhe (in Pixel) fest.
Text umbrechen	Ermöglicht, Text in den Zellen umzubrechen.
Sichtbar	Bestimmt, ob die Spalte in die Tabelle aufgenommen wird.
Schriftgröße	Legt die Schriftgröße in Punkt für den Text in der Spalte fest.
Art	Legt die Schriftart für den Text in der Spalte fest.
Farbe	Legt die Farbe der Spalte fest.
Fett	Zeigt den Text in der Spalte fett an.
Kursiv	Zeigt den Text in der Spalte kursiv an.

Siehe auch

To-Do-Listen verwenden (siehe Seite 82)

2.440 Tool-Palette

Ansicht>Tool-Palette

Verwenden Sie die Tool-Palette am Beginn eines neuen Projekts zum Hinzufügen von Komponenten zu einem Formular oder von Code-Snippets zum Quelltext-Editor.

Element	Beschreibung
Kategorien	Zeigt eine Liste mit Elementkategorien an und ermöglicht das Setzen der Tool-Palette auf eine bestimmte Kategorie.
	Legt den Filter für die Tool-Palette fest oder entfernt ihn. Beginnen Sie an einer beliebigen Stellen in der Tool-Palette mit der Eingabe des Namens desjenigen Elements, das Sie suchen. Die Tool-Palette zeigt automatisch nur die Elemente an, die mit den eingegebenen Buchstaben übereinstimmen. Klicken Sie das Filtersymbol an, um den Filter wieder zu entfernen.

Tip: Um die Kategorien oder die Elemente in der Tool-Palette neu anzuordnen, klicken Sie das Element oder die Kategorie an und legen Sie sie an der gewünschten Stelle in der Tool-Palette ab. Der Befehl Neuanordnung verhindert das Kontextmenü aktiviert/deaktiviert das Neuanordnen mittels Drag&Drop.

Kontextmenü

Klicken Sie die Tool-Palette mit der rechten Maustaste an, um die folgenden Befehle anzuzeigen.

Element	Beschreibung
Neue Kategorie hinzufügen	Zeigt das Dialogfeld Eine neue Kategorie erstellen an, in dem Sie eine leere Kategorie anlegen können. Anschließend können Sie Komponenten aus anderen Kategorien in die neue ziehen, um eine angepasste Kategorie zu erstellen.
Kategorie löschen	Löscht die ausgewählte Kategorie aus der Tool-Palette. Um die Kategorie wiederherzustellen, wählen Sie .NET-Komponenten anpassen.
Schaltfläche löschen	Löscht das ausgewählte Element aus der Tool-Palette.
Schaltfläche verbergen	Entfernt das ausgewählte Element aus der Tool-Palette, löscht es aber nicht.
Schaltfläche anzeigen	Zeigt zuvor mit Schaltfläche verbergen verborgene Elemente wieder an.
Installierte .NET-Komponenten	Zeigt das Dialogfeld Installierte .NET-Komponenten an, in dem Sie der Tool-Palette Komponenten hinzufügen können. Dieser Befehl steht nur zur Verfügung, wenn die Tool-Palette Komponenten enthält.
Alle Code-Snippets löschen	Entfernt benutzerdefinierte Codefragmente aus der Tool-Palette.
Kategorien automatisch ausblenden	Legt fest, dass jeweils nur eine Kategorie erweitert dargestellt wird.
Alle ausblenden	Zeigt nur die Kategorien der Elemente an.
Alle einblenden	Zeigt die Elemente aller Kategorien an.
Neuanordnung verhindern	Deaktiviert in der Tool-Palette die Neuanordnung der Kategorien durch Drag&Drop.
Palette zurücksetzen	Entfernt alle Anpassungen der Tool-Palette.
Eigenschaften	Zeigt die Seite Tool-Palette des Dialogfeldes Optionen an, auf der Sie das Erscheinungsbild der Tool-Palette ändern können.

Immer im Vordergrund	Zeigt die Tool Palette über allen anderen Fenstern des Desktops an.
Andockbar	Aktiviert die Drag&Dock-Funktion für die Tool-Palette.

2

Siehe auch

Der Tool-Palette Komponenten hinzufügen ( siehe Seite 70)

Komponenten in ein Formular einfügen ( siehe Seite 55)

Code-Snippets verwenden ( siehe Seite 50)

Elemente der Tool-Palette suchen ( siehe Seite 67)

2.441 Translation-Manager

Ansicht ▶ Translation-Manager

Verwenden Sie den Translation-Manager zum Anzeigen und Bearbeiten von Sprachressourcendateien.

Element	Beschreibung
Register Projekt	Zeigt die folgenden Register an: Das Register Sprachen enthält alle Sprachen im geöffneten Projekt sowie die lokale ID, die Dateierweiterung und das Übersetzungsverzeichnis für jede Sprache. Das Register Dateien enthält die Ressourcendateien für die im Register Sprachen ausgewählte Sprache. Zum Öffnen einer Ressourcendatei in einem Texteditor klicken Sie diese doppelt an.
Register Arbeitsbereich	Zeigt eine Baumhierarchie des Projekts an. Wenn Sie im linken Bereich einen Eintrag (keine Ressource) markieren, werden im rechten Bereich Zusammenfassungsinformationen dazu angezeigt. Wenn Sie eine Ressourcendatei im linken Bereich markieren, erscheint ein Gitter zum Anzeigen und Bearbeiten von Übersetzungen.

Kontextmenü Aktionen des Registers Arbeitsbereich

Das Kontextmenü Aktionen stellt einen schnellen Zugriff auf häufig benötigte Funktionen im Translation-Manager/Externen Translation-Manager bereit. Zum Öffnen des Menüs Aktionen können Sie im Register **Arbeitsbereich** entweder auf die Schaltfläche Aktionen klicken oder im Gitter mit der rechten Maustaste klicken.

Element	Beschreibung
Filter	Ermöglicht das Ein- und Ausblenden von Zeilen nach den folgenden Kriterien: Alle anzeigen, Umschalten (wechselt zwischen der Anzeige von im Kontextmenü Spalten ausgewählten bzw. nicht-ausgewählten Kriterien), Keine anzeigen, Nicht übersetzte anzeigen, Übersetzte anzeigen, Neu übersetzte anzeigen, Auto-übersetzte anzeigen und Nicht verwendete anzeigen (Übersetzungen, die auch nach dem Löschen der Ressource beibehalten wurden).
Spalten	Ermöglicht das Ein- oder Ausblenden von Spalten, indem Sie den jeweiligen Spaltennamen markieren. Alle anzeigen zeigt alle Spalten an, Umschalten wechselt zwischen der Anzeige von im Kontextmenü Spalten ausgewählten bzw. nicht-ausgewählten Kriterien und Keine anzeigen entfernt alle Spalten aus dem Anzeigegitter.
Bearbeiten	Zeigt das Dialogfeld Auswahl bearbeiten an, in dem Sie den Wert für die Zielsprache, den Status oder das Kommentarfeld bearbeiten können.
Ablage	Zeigt die folgenden Befehle an: Strings dem Wörterbuch hinzufügen speichert die Übersetzung der markierten Zeile(n) in der Wörterbuchdatenbank. Strings aus dem Wörterbuch holen durchsucht das Wörterbuch nach einer Übersetzung in der Zielsprache, deren Quell-String mit der markierten Ressource übereinstimmt.
Schrift	Zeigt das Dialogfeld Schriftart an, in dem Sie die Schriftart für Werte in der Basissprachenspalte, in der Zielsprachenspalte und der Kommentarspalte ändern können.
Vorherige kopieren	Zeile Überschreibt den ausgewählten Wert in der Zielsprachenspalte mit dem Wert der Zelle, die sich unmittelbar darüber befindet (nur wenn der Wert denselben Typ hat, wie z.B. eine Zahl oder ein Text-String).
Vorherige Übersetzung kopieren	Fügt eine frühere Version der Quell-Strings und ihrer Übersetzungen ein, die bei der Aktualisierung überschrieben wurden.
Original kopieren	Überschreibt den Zielsprachenwert mit dem Basissprachenwert und ändert den Status von Übersetzt in Nicht übersetzt.

Status zu Übersetzt ändern	Ändert die Spalte Status von Nicht übersetzt in Übersetzt. Sie können auch den Pfeil in den Dropdown-Feldern Nicht übersetzt/Übersetzt anklicken, um den Status zu ändern.
Nächster nicht übersetzter Begriff	Wechselt in die nächste Zeile im Gitter, die den Status Nicht übersetzt hat.
Alle auswählen	Markiert alle Werte in allen Zeilen und Spalten.

Tastenkürzel im Register Arbeitsbereich

Im Register Arbeitsbereich stehen die folgenden Tastenkürzel zur Verfügung:

Element	Beschreibung
STRG+A	Markiert alles im Gitter.
Strg+C	Kopiert die Auswahl in die Zwischenablage.
Strg+D	Kopiert die Übersetzung aus der vorherigen Zeile des Gitters.
Strg+E	Zeigt den Mehrzeilen-Editor an.
Strg+F	Zeigt das Dialogfeld Suchen an.
Strg+K	Behält das Formular im Vordergrund.
Strg+N	Sucht den nächsten nicht übersetzter Begriff.
Strg+O	Kopiert Text aus der Quellspalte (BasisSprache) in die Übersetzungsspalte.
Strg+P	Stellt die Übersetzung aus der Spalte Vorherige Übersetzung wieder her.
Strg+Q	Zeigt das Kontextmenü Aktionen an.
Strg + S	Speichert die Übersetzungen.
Strg+T	Ändert den Status in Übersetzt.
STRG+V	Fügt die Auswahl aus der Zwischenablage ein.
STRG+W	Setzt die Spaltenbreite zurück.
STRG+X	Schneidet die Auswahl aus und fügt sie in die Zwischenablage ein.
F6	Wechselt im Register Arbeitsbereich zwischen dem linken und rechten Ausschnitt.
Strg+F5	Aktualisiert das übersetzte Formular.
Strg+F7	Zeigt das Originalformular an.
Strg+F8	Zeigt das übersetzte Formular an.
Umschalt+Strg+F5	Aktualisiert das Gitter.
Umschalt+Strg+F7	Synchronisiert die übersetzte Version mit der BasisSprachenversion, z.B. wenn die Basisversion geändert wird.

Siehe auch

Anwendungen lokalisieren ([siehe Seite 1437](#))

Sprachen zu einem Projekt hinzufügen ([siehe Seite 86](#))

Ressourcendateien im Translation-Manager bearbeiten ([siehe Seite 88](#))

2.442 Mehrzeilen-Editor

Verwenden Sie den Mehrzeilen-Editor zum Bearbeiten von längeren Übersetzungen oder von mehrzeiligen Übersetzungen, die harte Zeilenumbrüche enthalten. Der Editor zeigt die Quell- und Zielsprachen in unterschiedlichen Bereichen an. Nur die Zielsprache kann bearbeitet werden.

Element	Beschreibung
Pfeilsymbole	Verschieben den vorherigen oder den nächsten String in das Gitter des Translation-Managers. Durch Anklicken der Schaltflächen werden Ihre Änderungen gespeichert.
Nebeneinander Untereinander	Ändert die Ausrichtung der Editorbereiche.
Speichern	Speichert Ihre Änderungen.
Schließen	Verwirft nicht-gespeicherte Änderungen und schließt den Editor.
Schrift	Ändert die Anzeigeschriftart im Editor. Die Änderungen beziehen sich auf den Bereich, in dem sich der Cursor befindet, wenn Sie die Schaltfläche Schriftart anklicken und wirken sich sowohl auf das Gitter des Translation-Managers als auch den Mehrzeilen-Editor aus.
Zeilenumbruch	Aktiviert oder deaktiviert den Umbruch von langen Zeilen.

Siehe auch

Anwendungen lokalisieren ( siehe Seite 1437)

Sprachen zu einem Projekt hinzufügen ( siehe Seite 86)

Ressourcendateien im Translation-Manager bearbeiten ( siehe Seite 88)

2.443 Typbibliothekseditor

Ansicht > Typbibliothek

Verwenden Sie dieses Dialogfeld, um Ihre Typbibliothek zu ändern. Der Editor erstellt automatisch die erforderliche IDL-Syntax. Alle vorgenommenen Änderungen werden in die entsprechende Implementierungsklasse übernommen (sofern sie mit einem Experten erstellt wurde).

Der Befehl **Ansicht > Typbibliothek** steht nur bei Projekten zur Verfügung, die eine Typbibliothek enthalten. Die Experten der Registerkarte ActiveX fügen dem Projekt automatisch eine Typbibliothek hinzu, wenn mit ihrer Hilfe ein COM-Objekt erstellt wird.

Objektliste

In der Objektliste werden alle Instanzen eines Informationstyps der aktuellen Typbibliothek in Form eines eindeutigen Symbols angezeigt. Wählen Sie ein Symbol aus, um seine Datenseiten im Informationsbereich auf der rechten Seite anzuzeigen.

Attribute (Registerkarte)

In dieser Registerkarte werden die Typinformationen für das Objekt angezeigt, das in der Objektliste markiert ist. Mithilfe der Eingabefelder können Sie die Werte ändern. Welche Attribute angezeigt werden, hängt vom ausgewählten Element ab.

Element	Beschreibung
Name	Ein Name, der die Typbibliothek beschreibt. Der Name darf keine Leer- oder Satzzeichen enthalten.
GUID	Der global eindeutige 128-Bit-Bezeichner für das Interface der Typbibliothek (ein Nachkomme von <code>ITypeLib</code>).
Version	Eine bestimmte Version der Typbibliothek in Situationen, in denen mehrere Versionen der Bibliothek vorhanden sind. Die Versionsangabe besteht entweder aus einem Paar von dezimalen Integern, die durch einen Punkt voneinander getrennt sind, oder aus einem einzelnen dezimalen Integer. Der erste Wert in einem Integerpaar steht für die Hauptversionsnummer, der zweite Wert gibt die Nebenversionsnummer an. Wenn nur ein Integer verwendet wird, ist dies die Hauptversionsnummer. Sowohl die Haupt- als auch die Nebenversionsnummer sind vorzeichenlose Integer des Typs <code>Short</code> im Bereich zwischen 0 und 65535.
LCID	Die Gebietsschema-ID der Sprache, die für alle Textstrings und Elemente der Typbibliothek verwendet wird.
Hilfe-String	Eine kurze Beschreibung der Typbibliothek. Das Attribut Hilfe-String wird in Kombination mit Hilfekontext verwendet, um Hilfeinformationen in Form einer Hilfedatei bereitzustellen. Der String wird bei der Erstellung der Hilfedatei dem Hilfekontext zugeordnet.
Hilfekontext	Die Hilfekontext-ID für die Hilfe der Typbibliothek. Sie bezeichnet das Thema innerhalb der Hilfedatei.
Hilfe-String-Kontext	Bei Hilfe-DLLs ist Hilfe-String-Kontext die Kontext-ID für die Hilfe der Typbibliothek. Sie wird zusammen mit dem Attribut Hilfe-String-DLL verwendet, um die Hilfe als eigene DLL bereitzustellen.
Hilfe-String-DLL	Der vollständige Name der Hilfe-DLL (falls vorhanden).
Hilfedatei	Der Name der Hilfedatei (.hlp), die mit der Typbibliothek verknüpft ist (falls vorhanden).

Anmerkung: Der Typbibliothekseditor unterstützt zwei Methoden für die Bereitstellung der Hilfe: Eine für die Bibliothek erstellte Windows-Standardhilfdatei (dies ist die gebräuchliche Methode) oder eine separate DLL mit Hilfeinformationen (diese Methode erleichtert die Lokalisierung). Die Hilfedatei, für die Sie die Hilfeattribute definiert haben, müssen Sie selbst bereitstellen.

Text (Registerkarte)

Diese Registerkarte enthält die Deklarationen des aktuell ausgewählten Elements in IDL oder Object Pascal. Sie können hier

Änderungen vornehmen und die Typinformationen überprüfen.

Alle Elemente einer Typbibliothek verfügen über die Registerkarte Text, auf der die IDL- oder Object Pascal-Syntax des Elements angezeigt wird. In der Registerkarte Typbibliothek des Dialogfeldes Umgebungsoptionen ist die Sprache festgelegt, die in der Registerkarte Text verwendet wird. Alle Änderungen, die Sie in anderen Registerkarten vornehmen, werden automatisch in die Registerkarte Text übernommen. Umgekehrt wirken sich Änderungen in der Registerkarte Text (beispielsweise bei der direkten Eingabe von IDL- oder Object Pascal-Quelltext) auf die anderen Registerkarten des Typbibliothekseditors aus.

Anmerkung: Wenn Sie IDL-Bezeichner verwenden, die der Typbibliothekseditor nicht unterstützt, führt dies zu einem Syntaxfehler. Gegenwärtig unterstützt der Editor nur IDL-Bezeichner, die sich auf die Typbibliothek beziehen. Bezeichner für RPCs werden nicht unterstützt.

Flags (Registerkarte)

In dieser Registerkarte werden Attribute angezeigt, mit der das in der Registerkarte Attribute beschriebene Objekt geändert werden kann. Diese Registerkarte steht nicht für alle Elemente zur Verfügung.

Einige Typbibliothekselemente verfügen über Flags, mit denen Sie bestimmte Merkmale oder Fähigkeiten aktivieren oder deaktivieren können. Mit den Kontrollkästchen der Registerkarte Flags können diese Flags ein- und ausgeschaltet werden.

Verwendet (Registerkarte)

Steht nur zur Verfügung, wenn die Typbibliothek in der Objektliste markiert ist. Hier werden weitere Typbibliotheken angezeigt, die Definitionen enthalten, von der die ausgewählte Bibliothek abhängig ist.

Zum Hinzufügen einer Abhängigkeit markieren Sie das Feld neben dem Namen der Typbibliothek. Die Definitionen in dieser Bibliothek können dann von der aktuellen Typbibliothek verwendet werden. Wenn Sie eine Bibliothek hinzufügen möchten, die nicht in der Liste steht, klicken Sie mit der rechten Maustaste und wählen Alle Typbibliotheken anzeigen.

Zum Entfernen einer Abhängigkeit entfernen Sie die Markierung von dem Feld neben dem Namen der Typbibliothek.

Um eine andere Typbibliothek anzuzeigen, markieren Sie diese, klicken mit der rechten Maustaste und wählen Typbibliothek anzeigen.

Implementierung (Registerkarte)

Steht nur zur Verfügung, wenn eine CoClass in der Objektliste markiert ist. Es wird eine Liste der Interfaces angezeigt, die von der CoClass implementiert werden. Sie können das dem Objekt zugeordnete Interface oder dessen Eigenschaften ändern.

Element	Beschreibung
Interface	Der Name eines Interface (oder eines DispInterface), das von der CoClass unterstützt wird. Dieser Name wird bei der Auswahl des Interface (bzw. des DispInterface) in der Registerkarte Attribute zugewiesen.
GUID	Der global eindeutige Bezeichner des Interface. Diese Angabe dient lediglich zu Informationszwecken und kann nicht geändert werden.
Dateiname	Gibt an, ob das Interface als Ereignisquelle fungiert. Wenn dies der Fall ist, wird das Interface nicht von der CoClass implementiert. Stattdessen implementieren die Clients das Interface, und die CoClass ruft die Clients über dieses Interface auf, wenn es Ereignisse auslöst.
Vorgabe	Gibt an, dass das Interface bzw. das DispInterface das Standard-Interface ist. Wenn eine Instanz der Klasse erstellt wird, gibt der Konstruktor standardmäßig dieses Interface zurück. Eine CoClass kann maximal zwei Standardelemente haben. Eines repräsentiert das primäre Interface, das andere ein optionales DispInterface, das als Ereignisquelle dient.
Restricted	Verhindert, dass das Element von Programmierer verwendet wird. Ein Interface kann entweder das Attribut Vorgabe oder Restricted haben.
VTable	Gibt an, ob Interface-Methoden unter Verwendung einer VTable aufgerufen werden können (im Gegensatz zu IDispatch-Aufrufen). Diese Angabe dient lediglich zu Informationszwecken und kann nicht geändert werden.

COM+ (Registerkarte)

Verwenden Sie diese Seite, um das Transaktionsattribut eines transaktionalen Objekts zu ändern, das mit MTS installiert werden soll, oder die COM+-Attribute einer CoClass zu modifizieren, die mit COM+ installiert werden soll.

Die Attribute dieser Registerkarte bestimmen außerdem, wie Automatisierungsobjekte, die nicht mit dem Experten für transaktionale Datenmodul-Objekte erstellt wurden, von der IDE in MTS-Packages oder COM+ Anwendungen installiert werden. Beachten Sie aber, dass solche Objekte nicht automatisch *IObjectControl* unterstützen. Das bedeutet, dass sie nicht über Aktivierungen oder Deaktivierungen benachrichtigt werden (sie unterstützen keine *OnActivate*- und *OnDeactivate*-Ereignisse). Da sie auch nicht über die Eigenschaft *ObjectContext* verfügen, muss der Objektkontext durch einen Aufruf der globalen Funktion *GetObjectContext* abgerufen werden.

Anmerkung: Bei der Installation in einem MTS-Package wird nur das Attribut *Transaktionsmodell* berücksichtigt. Alle anderen Einstellungen werden ignoriert. Wenn Sie beabsichtigen, das Objekt unter MTS zu installieren, müssen Sie ein Automatisierungsobjekt in einem In-Process-Server (DLL) verwenden.

Warnung: Die Attribute, die Sie in der Registerkarte COM+ festlegen, werden in der Typbibliothek als benutzerspezifische Daten codiert. Außerhalb von Delphi werden diese Daten nicht erkannt. Die Attributeneinstellungen sind daher nur wirksam, wenn Sie das transaktionale Objekt über die IDE installieren. Wenn das Objekt auf andere Weise weitergegeben werden soll, müssen Sie die Einstellungen explizit mit dem MTS-Explorer oder dem COM+ Komponenten-Manager festlegen.

Element	Beschreibung
Synchronisierung aufrufen	Nur COM+: Legt fest, auf welche Weise das Objekt an Aktivitäten beteiligt ist. Sie können so für eine zusätzliche Synchronisierung sorgen, die über die Standardsynchronisierung des Threading-Modells hinausgeht.
Transaktionsmodell	Legt das Transaktionsattribut fest, das angibt, wie das Objekt an Transaktionen teilnimmt. Die möglichen Werte für dieses Attribut hängen davon ab, ob das Objekt unter MTS oder unter COM+ eingesetzt wird. Wenn aufgrund dieser Einstellung Transaktionen unterstützt werden, muss auch das Attribut <i>Just In time-Aktivierung</i> aktiviert werden.
Objekt-Pooling	Nur COM+: Bestimmt, ob für Objektinstanzen Pooling gestattet ist. Wenn Sie Objekt-Pooling aktivieren, müssen Sie sicherstellen, dass das Objekt statuslos ist.
Zeitüberschreitung bei Erzeugung	Nur COM+: Legt fest, wie lange ein Objekt mit aktiviertem Pooling im Objektpool verbleibt, bevor es wieder freigegeben wird (in Millisekunden).
Inproc-Subskription ermöglichen	Ist nur wirksam, wenn die CoClass ein COM+ Ereignisobjekt ist. Dieses Attribut gibt an, ob In-Process-Anwendungen als Clients des Ereignisobjekts registriert werden können.
Parallel auslösen	Nur anwendbar, wenn die CoClass ein COM+-Ereignisobjekt darstellt. Legt fest, ob COM+ Ereignisse parallel (bei Multi-Threads) auslöst oder nach einander im selben Thread.

Parameter (Registerkarte)

Steht nur zur Verfügung, wenn eine Eigenschaft oder Methode in der Objektliste markiert ist. Sie können hier die Parameter und den Rückgabewert von Methoden festlegen (dies gilt auch für Methoden mit Eigenschaftszugriff).

Element	Beschreibung
Name	Gibt den Namen des Parameters an. Sie können diesen Wert direkt bearbeiten.
Typ	Gibt den Datentyp des Parameters an. Wählen Sie einen Typ aus der Dropdown-Liste der verfügbaren Typen, die angezeigt wird, wenn Sie in der Spalte Typ klicken.

Vorgabewert	<p>Geben Sie den Vorgabewert für einen optionalen Parameter direkt in diese Spalte ein. Alle nachfolgenden Parameter müssen optional sein. Für vorhergehende Parameter sollte ebenfalls ein Vorgabewert definiert sein.</p> <p>Wenn Sie in Object Pascal arbeiten, werden lokale IDs mit Hilfe des Parametertyp-Bezeichners <i>TLCID</i> angegeben. In IDL erfolgt diese Angabe durch einen Parametermodifizierer.</p>
-------------	--

2.444 Formular anzeigen

Ansicht▶Formulare

Verwenden Sie dieses Dialogfeld, um die Formulare des aktuellen Projektes schnell anzuzeigen. Wenn Sie ein Formular markieren, wird es zum aktiven Formular und die damit verbundene Unit wird zum aktiven Modul im Quelltext-Editor.

2.445 Units anzeigen

2

Ansicht > Units

Verwenden Sie dieses Dialogfeld, um die Projektdatei oder eine Unit des aktuellen Projekts anzuzeigen. Wenn Sie eine Unit wählen, erscheint die aktive Seite im Quelltext-Editor.

2.446 Fensterliste

Ansicht ▾ Fensterliste

Verwenden Sie dieses Dialogfeld zum Anzeigen einer Liste der geöffneten Fenster.

Element	Beschreibung
Windows	Markieren Sie ein Fenster und klicken Sie OK an, um dieses Fenster anzuzeigen.

2.447 Assemblierungs-Metadaten-Explorer

Datei ▶ Öffnen

Verwenden Sie den Assemblierungs-Metadaten-Explorer zum Untersuchen von in einer .NET-Assemblierung enthaltenen Typen.

Symbol	Typ	Verfügbare Register
	Assemblierung	Eigenschaften, Attribute, Flags, Verwendet
	Namespace	Properties
	Klasse	Eigenschaften, Attribute, Flags, Implementierungen
	Sealed-Klasse	Eigenschaften, Attribute, Flags, Implementierungen
	Interface	Eigenschaften, Attribute, Flags, Implementierungen, Implementierung
	Methode	Eigenschaften, Attribute, Flags, Parameter, Aufruf-Graph
	Methode mit Rückgabewert	Eigenschaften, Attribute, Flags, Parameter, Aufruf-Graph
	Eigenschaft mit Getter und Setter	Eigenschaften, Flags
	Eigenschafts-Getter-Methode	Eigenschaften, Flags
	Eigenschafts-Setter-Methode	Eigenschaften, Flags
	Feld	Eigenschaften, Flags
	Ereignis	Eigenschaften, Attribute, Flags

Die auf den Registern angezeigten Metadaten-Elemente sind je nach Typ des ausgewählten Elements unterschiedlich. In den folgenden Abschnitten sind alle auf den Registern angezeigten Metadaten-Elemente aufgeführt.

Register Eigenschaften

Zeigt die Eigenschaften des ausgewählten Elements an.

Element	Anwendbar für die Typen	Bemerkungen
Name	Alle	
GUID	Assemblierung	
Version	Assemblierung	
Kultur	Assemblierung	
Revision	Assemblierung	
Build-Nummer	Assemblierung	
Namespace	Klasse	
Assemblierung	Klasse	
ID	Klasse, Feld, Eigenschaft, Methode, Ereignis	Die ID ist eine interne Zahl, die angibt, wo der Typ in den internen Metadatentabellen der Assemblierung zu finden ist.

Erweitert	Klasse	Die Basisklasse der ausgewählten Klasse
Erweitert ID	Klasse	Die interne ID der Basisklasse
Werttyp	Feld	
Wert	Feld	
Rückgabetyp	Methode	

Register Attribute

Das Register Attribute zeigt alle Attribute (einschließlich benutzerdefinierter Attribute) an, die dem ausgewählten Element im Quelltext zugewiesen wurden. Bei allen Attributen wird der Name und der Attributwert angezeigt.

Register Flags

Das Register Flags zeigt die Metadaten-Flags an, die für das markierte Element anwendbar sind. Jedes Flag wird durch ein Kontrollkästchen dargestellt. Wenn das Kontrollkästchen markiert ist, ist das Flag in den Metadaten des markierten Elements gesetzt. Ist das Kontrollkästchen nicht markiert, wurde das Flag dem markierten Element nicht zugewiesen.

Register Verwendet

Das Register Verwendet zeigt die Liste der Assemblierungen an, von welchen die ausgewählte Assemblierung abhängig ist. Jede aufgeführte Assemblierung muss auf dem Rechner des Endbenutzers vorhanden sein.

Register Implementierungen

Das Register Implementierungen steht nur zur Verfügung, wenn das markierte Element eine Klasse, eine Sealed-Klasse oder ein Interface ist. Dieses Register enthält alle Interfaces, die von dem ausgewählten Element implementiert werden. Alle implementierten Interfaces sind Links, die Sie anklicken können. Durch das Anklicken wird das Element in der Hierarchie markiert und seine Metadaten-Eigenschaften werden angezeigt. Mit den Schaltflächen Vorwärts und Zurück in der Symbolleiste können Sie schnell zurück zu der vorher ausgewählten Klasse oder Interface navigieren.

Register Implementierung

Das Register Implementierung ist nur sichtbar, wenn ein Interface im linken Bereich markiert ist. Das Register zeigt alle Klassen an, die das Interface implementieren.

Register Parameter

Das Register Parameter ist nur sichtbar, wenn eine Methode im linken Bereich markiert ist. Alle Parameter werden mit Namen, Typ und Modifizierer (wie z.B. **ref** und **out**) angezeigt.

Register Aufruf-Graph

Das Register Aufruf-Graph ist nur sichtbar, wenn eine Methode im linken Bereich markiert ist. Dieses Register ist in zwei Bereiche unterteilt: Der obere Bereich zeigt die Methoden an, die die ausgewählte Methode aufrufen. Der untere Bereich zeigt alle Methoden an, die von der ausgewählten Methode aufgerufen werden.

Bestimmte Methoden, durch eine blaue Farbe und eine Unterstreichung gekennzeichnet, sind anklickbare Links; diese Methoden befinden sich in der aktuellen Assemblierung. Klicken Sie einen Methoden-Link an, damit diese Methode im linken Bereich markiert wird. Andere Methoden, die in den Bereichen Ruft auf und Aufgerufen von aufgeführt sind, sind keine Links; diese Methoden sind nicht in der aktuellen Assemblierung definiert. Sie können mit den Navigations-Schaltflächen in der Symbolleiste vorwärts und zurück zu den vorher ausgewählten Elementen navigieren.

Siehe auch

[COM Interop in verwalteten Anwendungen](#)

[Referenzen zu einem COM-Server hinzufügen](#)

[ActiveX-Steuerelemente zur Tool-Palette hinzufügen](#)

2.448 Typbibliothek-Explorer

Datei ▶ Öffnen

Verwenden Sie den Typbibliothek-Explorer zum Untersuchen von Typen und Interfaces, die in einer Windows-Typbibliothek definiert sind.

--	--

Typbibliothekselement	Symbol	Seite der Typinformationen	Inhalt der Seite
Typbibliothek	❖	Attribute	Name, Version, GUID und registrierter Speicherort der Typbibliothek. Zeigt auch den Hilfekontext und Informationen zur Hilfedatei an.
		Verwendet	Eine Tabelle mit den Namen und GUIDs von abhängigen Typbibliotheken.
		Flags	Flags, die festlegen, wie andere Anwendungen die Typbibliothek verwenden können.
CoClass	●	Attribute	Name, GUID, Hilfekontext und Informationen zur Hilfedatei..
		Flags	Flags, die angeben, wie Clients Instanzen erzeugen und verwenden können, ob die CoClass in einem Browser sichtbar ist, ob die CoClass ein ActiveX-Steuerelement ist und ob die Elemente der CoClass zusammengefasst werden können.
		Implementierung	Eine Tabelle mit den Interfaces (und deren Attributen), die die Klasse implementiert.
Interface	🔑	Attribute	Name, Version, GUID, Hilfekontext und Informationen zur Hilfedatei..
		Flags	Flags, die angeben, ob das Interface verborgen, dual, automatisierungskompatibel und/oder erweiterbar ist.
DispInterface	🔑	Attribute	Name, Version, GUID, Hilfekontext und Informationen zur Hilfedatei..
		Flags	Flags, die angeben, ob das DispInterface verborgen, dual, automatisierungskompatibel und/oder erweiterbar ist.
Methode Methode Rückgabewert mit	👉	Attribute	Name, Dispatch-ID, Vtable-Offset, Hilfe-String und Hilfekontextinformationen.
		Flags	Flags, die angeben, wie Clients die Methode anzeigen und verwenden können, ob es sich um eine Standardmethode für das Interface handelt und ob die Methode ersetzt werden kann.
		Parameter	Eine Tabelle mit den Namen und Typen aller Parameter der Methode und - falls anwendbar - der Rückgabewert.

SetByRef-Methode		Attribute	ToDo
Getter-Methode Setter-Methode		Attribute	Name, Dispatch-ID, Vtable-Offset, Hilfe-String und Hilfekontextinformationen.
		Flags	Flags, die angeben, wie der Client die Methode anzeigen und verwenden kann und ob sie verborgen und/oder suchbar ist.
		Parameter	Eine Tabelle mit den Namen und Typen aller Parameter in der Methode.
DispProperty		Attribute	Name, Hilfe-String und Hilfekontextinformationen.
		Flags	Flags, die angeben, wie Clients die Eigenschaft anzeigen und verwenden können, ob es eine Standardeigenschaft für das Interface ist, usw.
Alias		Attribute	Name, Version, GUID, der Typ, den der Alias repräsentiert und Hilfekontextinformationen.
Record		Attribute	Name, Version, GUID, Hilfe-String und Hilfekontextinformationen.
Union		Attribute	Name, Version, GUID, Hilfe-String und Hilfekontextinformationen.
Aufzählung		Attribute	Name, Version, GUID, Hilfe-String und Hilfekontextinformationen.
Modul		Attribute	Name, Version, GUID, Name der zugeordneten DLL, Hilfe-String und Hilfekontextinformationen.
Feld		Attribute	Name, Typinformationen, Hilfe-String und Hilfekontextinformationen.
		Flags	Flags, die angeben, wie Clients das Feld anzeigen und verwenden können, ob das Feld einen Standardwert hat und ob es bindbar ist.
Konstante		Attribute	Name, Wert, Typ (für Modulkonstanten), Hilfe-String und Hilfekontextinformationen.
		Flags	Flags, die angeben, wie Clients die Konstante anzeigen und verwenden können, ob sie einen Standardwert repräsentiert und ob sie bindbar ist.

Siehe auch

[COM Interop in verwalteten Anwendungen](#)

[Referenzen zu einem COM-Server hinzufügen](#)

[ActiveX-Steuerelemente zur Tool-Palette hinzufügen](#)

2.449 Suchen

Verwenden Sie dieses Dialogfeld zur Suche nach Elementarten (wie Klassen oder Interfaces) oder zur Suche nach einem bestimmten Element in einer .NET-Assemblierung oder einer Typbibliothek.

Element	Beschreibung
Suchen nach	Der Textstring, nach dem in der Assemblierung oder Typbibliothek gesucht werden soll.
Typ	Eine Dropdown-Liste mit den Elementtypen, nach denen gesucht werden kann. Dies sind: Klasse, Interface, Methode, Eigenschaft, Feld und Ereignis. Wenn Sie keinen Typ angeben, werden alle Elemente gefunden, die mit dem Suchstring übereinstimmen.
Groß-/Kleinschreibung	Wenn diese Option aktiviert ist, wird bei der Suche die Groß-/Kleinschreibung berücksichtigt. Ist die Option deaktiviert, werden alle mit dem Suchstring übereinstimmenden Elemente ohne Berücksichtigung der Schreibweise gefunden.
Suchen	Klicken Sie hier, um den Suchvorgang zu starten.

Elemente, die mit den Suchkriterien übereinstimmen, werden in einer Tabelle im Dialogfeld Suchen angezeigt. Bei den Elementen in dieser Tabelle handelt es sich um Links, die durch Klicken aktiviert werden. Wenn Sie auf ein Element klicken, wird es im linken Bereich der Explorer-Ansicht ausgewählt.

Tip: Das Dialogfeld Suchen ist ein nicht-modales Fenster. Es kann geöffnet bleiben, während Sie im Hauptfenster des Explorers weiterarbeiten.

Siehe auch

[COM Interop in verwalteten Anwendungen](#)

[Referenzen zu einem COM-Server hinzufügen](#)

[ActiveX-Steuerelemente zur Tool-Palette hinzufügen](#)

3 Quelltexthaltepunkt hinzufügen

[Start](#) ▶ [Haltepunkt hinzufügen](#) ▶ [Quelltexthaltepunkt](#)

Verwenden Sie dieses Dialogfeld zum Festlegen eines Haltepunktes in einer Zeile Ihres Quelltextes oder zum Ändern der Eigenschaften eines vorhandenen Haltepunktes. Je nachdem, von wo aus das Dialogfeld aufgerufen wird, kann es auch den Titel Eigenschaft des Haltepunkts oder Eigenschaften des Adresshaltepunkts haben.

Element	Beschreibung
Dateiname	Gibt die Quelltextdatei für den Quelltexthaltepunkt an. Geben Sie den Namen einer Quelltextdatei für den Haltepunkt ein.
Zeilennummer	Setzt oder ändert die Zeilennummer für den Haltepunkt. Geben Sie die Zeilennummer für den Haltepunkt ein oder ändern Sie diese.
Bedingung	Legt einen Bedingungsausdruck fest, der bei jedem Durchlauf ausgewertet wird. Die Programmausführung wird angehalten, wenn der Ausdruck true ergibt. Geben Sie einen Bedingungsausdruck zum Beenden der Programmausführung ein. Sie können jeden gültigen Sprachausdruck verwenden. Alle Symbole im Ausdruck müssen jedoch vom Haltepunkt aus erreichbar sein. Funktionen sind zulässig, wenn sie einen Booleschen Wert zurückgeben.
Durchlaufzähler	Hält die Programmausführung an einer bestimmten Zeilennummer nach einer gegebenen Anzahl von Durchläufen an. Geben Sie die Anzahl der Durchläufe an. Der Debugger inkrementiert die Durchlaufzählung jedes Mal, wenn der Haltepunkt erreicht wird. Wenn der Durchlaufzähler die festgelegte Zahl erreicht hat, hält der Debugger die Programmausführung an. Wenn Sie beispielsweise 3 Durchläufe angeben, werden im Zähler die Werte 0 von 3, 1 von 3, 2 von 3 und 3 von 3 angezeigt. Die Programmausführung wird bei 3 von 3 angehalten. Da der Debugger den Zähler bei jedem Durchlauf inkrementiert, können Sie diese verwenden, festzustellen, welcher Schleifendurchlauf fehlschlägt. Setzen Sie den Durchlaufzähler auf die maximale Anzahl von Schleifendurchläufen und starten Sie Ihr Programm. Schlägt das Programm fehl, können Sie die Anzahl der Schleifendurchläufe berechnen, indem Sie die Anzahl der ausgeführten Durchläufe prüfen. Wenn Sie Durchlaufzähler mit Bedingungen verwenden, pausiert die Programmausführung, wenn der Bedingungsausdruck zum <i>n</i> -ten Mal true ergibt. Der Debugger dekrementiert den Durchlaufzähler nur dann, wenn der Bedingungsausdruck true ist.

Gruppe	Erstellt eine Haltepunktgruppe, in die der Haltepunkt aufgenommen wird. Um eine bereits vorhandene Gruppe zu verwenden, wählen Sie sie aus der Dropdown-Liste aus. Haltepunktgruppen ermöglichen das Ausführen derselben Aktionen für alle Haltepunkte in der Gruppe.
Erweitert	Erweitert das Dialogfeld mit Feldern zum Zuordnen von Aktionen zu Haltepunkten
Break	Hält die Ausführung an. Dies ist die herkömmliche Standardaktion eines Haltepunkts.
Spätere Exceptions ignorieren	Ignoriert alle nachfolgenden Exceptions, die vom aktiven Prozess in der aktuellen Debugging-Sitzung ausgelöst werden (der Debugger hält nicht bei jeder Exception an). Verwenden Sie diese und die Option Spätere Exceptions behandeln paarweise. Sie können bestimmte Anweisungsblöcke in ein <i>Ignorieren / Behandeln</i> -Paar einschließen und dadurch verhindern, dass der Debugger bei Exceptions in diesem Quelltext anhält.
Spätere Exceptions behandeln	Behandelt alle nachfolgenden Exceptions, die vom aktiven Prozess in der aktuellen Debugging-Sitzung ausgelöst werden (der Debugger hält dann entsprechend der Einstellungen in Tools ▸ Optionen ▸ Debugger-Optionen ▸ Sprach-Exceptions an). Diese Option hält bei allen Exceptions an. Mit ihr können Sie das normale Verhalten wiederherstellen, wenn die Exception-Behandlung zuvor mit Spätere Exceptions ignorieren ausgeschaltet wurde.
Meldung protokollieren	Schreibt die angegebene Meldung in das Ereignisprotokoll. Sie können den Wortlaut dieser Meldung eingeben.
Eval-Ausdruck	Wertet den angegebenen Ausdruck aus und schreibt - da die Option Ergebnis protokollieren standardmäßig aktiviert ist - das Ergebnis der Auswertung in das Ereignisprotokoll. Deaktivieren Sie Ergebnis protokollieren, wenn die Auswertung ohne Protokollierung durchgeführt werden soll.
Ergebnis protokollieren	Schreibt das Ergebnis der Auswertung in das Ereignisprotokoll, wenn Text in das Feld Eval-Ausdruck eingegeben wird. Bei deaktivierter Option findet keine Protokollierung statt.
Gruppe aktivieren	Aktiviert alle Haltepunkte der angegebenen Gruppe. Wählen Sie den gewünschten Gruppennamen aus.
Gruppe deaktivieren	Deaktiviert alle Haltepunkte der angegebenen Gruppe. Wählen Sie den gewünschten Gruppennamen aus.
Aufruf-Stack protokollieren	Zeigt den gesamten oder einen Teil des Aufruf-Stack im Fenster Ereignisprotokoll an, wenn ein Haltepunkt erreicht wird. Gesamter Stack zeigt den gesamten Aufruf-Stack an. Teil-Stack zeigt nur die Anzahl der im Feld Anzahl der Frames festgelegten Frames an.

4 Reference

4.1 Delphi-Referenz

Dieser Abschnitt beschreibt die Delphi-Sprache, die Delphi-Compiler-Direktiven und -Fehler, die in Delphi-Quelltext auftreten können.

4.1.1 Delphi-Sprachreferenz

Die Sprachreferenz enthält eine Beschreibung der Sprache Delphi und ihrer Verwendung in Entwicklungstools von CodeGear. Dabei wird sowohl die Win32- als auch die .NET-Plattform berücksichtigt. Auf sprachspezifische Unterschiede zwischen den beiden Plattformen wird gesondert hingewiesen.

4.1.1.1 Übersicht über Delphi

In diesem Kapitel finden Sie eine kurze Einführung in Delphi-Programme und Erläuterungen zur Programmorganisation.

4.1.1.1.1 Sprach-Übersicht

Delphi ist eine höhere Compiler-Sprache mit strenger Typisierung, die eine strukturierte und objektorientierte Programmierung ermöglicht. Sie basiert auf Objekt Pascal. Zu ihren Vorteilen gehören einfache Lesbarkeit, schnelle Compilierung und modulare Programmierung durch Verteilen des Quelltextes auf mehrere Unit-Dateien. Delphi verfügt über spezielle Leistungsmerkmale, die das Komponentenmodell und die visuelle Entwicklungsumgebung von CodeGear unterstützen. In den Beschreibungen und Beispielen dieser Sprachreferenz wird davon ausgegangen, dass Sie die CodeGear-Entwicklungstools verwenden.

Die meisten Entwickler, die mit Software-Entwicklungstools von CodeGear arbeiten, schreiben und compilieren ihre Programme in der integrierten Entwicklungsumgebung (IDE). Hier kümmert sich das Entwicklungstool um viele Details, die beim Einrichten von Projekten und Quelltextdateien von Bedeutung sind (z. B. Verwalten der Abhängigkeiten zwischen den Units). Dabei gelten Regeln bezüglich der Programmorganisation, die aber nicht zur Sprachdefinition von Object Pascal gehören. So werden beispielsweise bestimmte Namenskonventionen für Dateien und Programme verwendet, an die Sie aber nicht gebunden sind, wenn Sie Programme außerhalb der IDE schreiben und in der Befehlszeile compilieren.

Im allgemeinen geht diese Sprachreferenz davon aus, dass Sie in der IDE arbeiten und Anwendungen mithilfe der CodeGear-VCL (Visual Component Library) erstellen. Wo es erforderlich ist, wird jedoch zwischen Delphi-spezifischen Regeln und der normalen Programmierung mit Object Pascal unterschieden. Beschrieben werden die Delphi-Compiler für Win32 und für die .NET-Plattform. Auf plattformspezifische Unterschiede und Funktionen wird bei Bedarf hingewiesen.

Dieser Abschnitt enthält Informationen zu folgenden Themen:

- Programmorganisation. Eine Beschreibung der grundlegenden Leistungsmerkmale der Sprache, die das Aufteilen einer Anwendung in Units und Namespaces ermöglichen.
- Beispielprogramme. Kurze Beispiele für Konsolen- und GUI-Anwendungen veranschaulichen die Verwendung des Befehlszeilen-Compilers.

Programmorganisation

Delphi-Programme werden normalerweise auf mehrere Quelltextmodule (so genannte Units) verteilt. Die meisten Programme beginnen mit einer **program**-Kopfzeile, in der der Programmname angegeben wird. Darauf folgt eine optionale **uses**-Klausel und ein Block von Deklarationen und Anweisungen. Mit der **uses**-Klausel geben Sie die Units an, die in das Programm eingebunden

werden. Units können von mehreren Programmen gemeinsam benutzt werden und verfügen häufig über eigene **uses**-Klauseln. Die **uses**-Klausel liefert dem Compiler Informationen über die Modulabhängigkeiten. Da diese Abhängigkeiten in den Modulen selbst gespeichert werden, benötigen die meisten Delphi-Programme keine Make-Dateien, Header-Dateien oder **#include**-Präprozessordirektiven.

Delphi-Quelltextdateien

Der Compiler erwartet Delphi-Quelltexte in einem der folgenden Dateitypen:

- Unit-Quelltextdateien mit der Namenserweiterung .PAS
- Projektdateien mit der Namenserweiterung .DPR
- Package-Quelltextdateien mit der Namenserweiterung .DPK

In den Unit-Quelltextdateien befindet sich in der Regel der größte Teil des Programmcodes. Jede Anwendung besteht aus einer Projektdatei und mehreren Unit-Dateien. Die Projektdatei entspricht dem Hauptprogramm im herkömmlichen Pascal und organisiert die Units der Anwendung. CodeGear-Entwicklungstools verwalten für jede Anwendung automatisch eine Projektdatei.

Wenn Sie ein Programm in der Befehlszeile compilieren, können Sie den gesamten Quelltext in Unit-Dateien (.PAS) aufnehmen. Erstellen Sie die Anwendung jedoch mithilfe der IDE, wird eine Projektdatei (.DPR) angelegt.

Package-Quelltextdateien ähneln Projektdateien, werden aber für spezielle, dynamisch ladbare Bibliotheken (so genannte Packages) verwendet.

Weitere Anwendungsdateien

CodeGear-Produkte verwenden neben den Quelltextmodulen auch die folgenden Dateitypen, die keinen Pascal-Code enthalten und ebenfalls automatisch von der IDE verwaltet werden:

- VCL-Formulardateien mit der Namenserweiterung .DFM (Win32) oder .NFM (.NET)
- Ressourcendateien mit der Namenserweiterung .RES
- Projektdateien mit der Namenserweiterung .DOF

Eine VCL-Formulardatei enthält die Beschreibung der Eigenschaften des Formulars und seiner Komponenten. Sie repräsentiert ein Formular, das normalerweise einem Fenster oder Dialogfeld in einer Anwendung entspricht. Formulardateien können in der IDE als Text angezeigt und bearbeitet und danach entweder im Text- oder im Binärformat gespeichert werden. (Das Textformat eignet sich hervorragend zur Versionskontrolle.) Obwohl Formulardateien standardmäßig als Text gespeichert werden, ist eine manuelle Bearbeitung nicht üblich. Normalerweise werden dazu die visuellen Tools von CodeGear verwendet. Jedes Projekt hat mindestens ein Formular. Zu jedem Formular gehört eine Unit-Datei (.PAS), die standardmäßig denselben Namen wie die Formulardatei erhält.

Zusätzlich zu den VCL-Formulardateien verwendet jedes Projekt eine Ressourcendatei (.RES), in der das Anwendungssymbol und andere Ressourcen, wie Strings, gespeichert werden. Diese Datei erhält automatisch denselben Namen wie die Projektdatei (.DPR).

Die Projektoptionsdatei (.DOF) enthält Compiler- und Linker-Einstellungen, Suchpfadinformationen, Versionsinformationen usw. Diese Datei erhält automatisch denselben Namen wie die Projektdatei (.DPR). Die Einstellungen in dieser Datei können im Dialogfeld *Projektoptionen* festgelegt werden.

Verschiedene Tools in der IDE speichern Daten in speziellen Dateitypen. Desktop-Konfigurationsdateien (.DSK) enthalten Informationen über die Anordnung der Fenster und andere Konfigurationseinstellungen. Desktop-Einstellungen können projektspezifisch sein oder für die gesamte Umgebung gelten. Die Desktop-Konfigurationsdateien haben keine direkten Auswirkungen auf die Compilierung.

Vom Compiler generierte Dateien

Wenn Sie eine Anwendung oder ein Package zum ersten Mal erstellen, generiert der Compiler für jede im Projekt verwendete

Unit eine kompilierte Unit-Datei (.DCU für Win32 und .DCUIL für .NET). Diese Dateien werden dann zu einer ausführbaren Datei bzw. zu einem gemeinsam genutzten Package gelinkt. Bei der ersten Erstellung eines Package wird für jede im Package enthaltene neue Unit eine Datei erstellt. Danach generiert der Compiler eine .DCP- und eine Package-Datei. Wenn der Befehlszeilschalter **GD** verwendet wird, generiert der Linker eine Map- und eine .DRC-Datei. Die .DRC-Datei enthält String-Ressourcen und kann in eine Ressourcendatei kompiliert werden.

Beim Erstellen eines Projekts wird eine Unit nur dann erneut kompiliert, wenn ihre Quelltextdatei (.PAS) seit dem letzten Compilieren geändert wurde, die zugehörige .DCU/.DCP-Datei nicht gefunden werden kann, der Compiler explizit dazu angewiesen wird oder das Interface der Unit von einer anderen Unit abhängig ist, die geändert wurde. Wenn der Compiler auf die entsprechende kompilierte Unit-Datei zugreifen kann und die Unit nicht von anderen Units abhängig ist, die geändert wurden, wird die Quelltextdatei der Unit nicht benötigt.

Beispielprogramme

Die folgenden Beispielprogramme zeigen die Grundzüge der Programmierung mit Delphi. Es handelt sich um einfache Anwendungen, die nicht in der IDE, sondern nur in der Befehlszeile kompiliert werden können.

Eine einfache Konsolenanwendung

Das folgende Beispiel zeigt eine einfache Konsolenanwendung, die Sie in der Befehlszeile compilieren und ausführen können.

```
program greeting;

{$APPTYPE CONSOLE}

var MyMessage: string;

begin
  MyMessage := 'Hello world!';
  Writeln(MyMessage);
end.
```

Durch die erste Anweisung wird das Programm *Greeting* deklariert. Die Direktive `{$APPTYPE CONSOLE}` teilt dem Compiler mit, dass es sich um eine Konsolenanwendung handelt, die in der Befehlszeile ausgeführt wird. In der nächsten Zeile wird die String-Variable *MyMessage* deklariert (in Delphi gibt es echte String-Datentypen). Im Programmblock wird der Variable *MyMessage* der String "Hello world!" zugewiesen. Anschließend wird die Variable mit der Standardprozedur *Writeln* an die Standardausgabe gesendet. (*Writeln* ist implizit in der Unit *System* definiert, die automatisch in jede Anwendung eingebunden wird).

Sie können nun das Programm in eine Datei mit dem Namen `greeting.pas` oder `greeting.dpr` eingeben und dann compilieren:

```
dcc32 greeting
```

Hiermit erhalten Sie eine ausführbare Win32-Datei. Mit der Anweisung

```
ccil greeting
```

wird eine ausführbare Datei für die .NET-Plattform erstellt. In beiden Fällen gibt die ausführbare Datei die Meldung `Hello world!` auf dem Bildschirm aus.

Neben seiner Einfachheit unterscheidet sich dieses Beispiel noch in anderen wichtigen Punkten von den Anwendungen, die Sie normalerweise mit CodeGear-Entwicklungstools erstellen. Zum einen handelt es sich bei dem Beispiel um eine Konsolenanwendung. Mit CodeGear-Entwicklungstools werden hauptsächlich Anwendungen mit grafischen Benutzeroberflächen entwickelt, in denen natürlich keine *Writeln*-Aufrufe benötigt werden. Außerdem befindet sich das gesamte Beispielprogramm in einer einzigen Datei. Bei einer typischen GUI-Anwendung steht der Programmkopf (im Beispiel die erste Zeile) in einer separaten Projektdatei, die mit Ausnahme einiger Routinenaufrufe keinerlei Programmlogik enthält.

Ein komplexeres Beispiel

Das Programm im nächsten Beispiel ist auf zwei Dateien verteilt, eine Projektdatei und eine Unit-Datei. Die Projektdatei, die Sie

unter dem Namen `greeting.dpr` speichern können, hat folgenden Inhalt:

```
program greeting;

{$APPTYPE CONSOLE}

uses Unit1;

begin
  PrintMessage('Hello World!');
end.
```

In der ersten Zeile wird das Programm `greeting` deklariert, das wiederum eine Konsolenanwendung ist. Über die Klausel `uses Unit1;` wird dem Compiler mitgeteilt, dass das Programm `greeting` von einer Unit namens `Unit1` abhängig ist. Zum Schluss wird die Prozedur `PrintMessage` mit dem String `Hello World!` aufgerufen. Die Prozedur `PrintMessage` ist in `Unit1` definiert. `Unit1` hat folgenden Inhalt (speichern Sie sie unter dem Namen `Unit1.pas`):

```
unit Unit1;

interface

procedure PrintMessage(msg: string);

implementation

procedure PrintMessage(msg: string);
begin
  Writeln(msg);
end;

end.
```

`Unit1` definiert eine Prozedur namens `PrintMessage`, die einen String als Argument übernimmt und diesen String an die Standardausgabe sendet. (In Delphi heißen Routinen ohne Rückgabewert *Prozeduren*. Routinen, die einen Wert zurückgeben, heißen *Funktionen*.) Beachten Sie, dass `PrintMessage` zweimal in `Unit1` deklariert ist. Die erste Deklaration im `interface`-Abschnitt macht `PrintMessage` für andere Module verfügbar (z. B. für das Programm `greeting`), die `Unit1` einbinden. Durch die zweite Deklaration (im `implementation`-Abschnitt) erfolgt die eigentliche Definition der Prozedur `PrintMessage`.

Sie können `greeting` nun in der Befehlszeile compilieren:

```
dcc32 greeting
```

Hiermit erhalten Sie eine ausführbare Win32-Datei. Mit der Anweisung

```
ccil greeting
```

wird eine ausführbare Datei für die .NET-Plattform erstellt.

`Unit1` braucht in der Befehlszeile nicht angegeben zu werden. Der Compiler liest beim Bearbeiten von `greeting.dpr` automatisch die Unit-Dateien ein, von denen `greeting` abhängig ist. Die neue ausführbare Datei führt dieselbe Operation durch wie im ersten Beispiel, gibt also die Meldung `Hello world!` aus.

Eine VCL-Anwendung

Als nächstes erstellen wir eine Anwendung mithilfe von VCL-Komponenten (VCL, Visual Component Library) in der IDE. Im Programm werden die automatisch generierten Formular- und Ressourcendateien verwendet. Es kann daher nicht allein aus dem Quelltext compiliert werden. Das Beispiel zeigt einige wichtige Merkmale von Delphi. Neben mehreren Units werden auch Klassen und Objekte (siehe Seite 948) verwendet.

Die Anwendung besteht aus einer Projektdatei und zwei neuen Unit-Dateien. Betrachten wir zuerst die Projektdatei:

```
program greeting;
```

```

uses Forms, Unit1, Unit2;
{$R *.res} // Diese Direktive linkt die Ressourcendatei des Projekts.

begin
  // Aufruf der globalen Application-Instanz Application
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.CreateForm(TForm2, Form2);
  Application.Run;
end.

```

Unser Programm heißt auch diesmal greeting. In die **uses**-Klausel werden drei Units eingebunden: **Forms** ist ein Bestandteil der VCL, **Unit1** gehört zum Hauptformular der Anwendung (**Form1**) und **Unit2** zu einem weiteren Formular (**Form2**).

In Programm finden mehrere Aufrufe des Objekts **Application** statt, das eine Instanz der in **Forms** deklarierten Klasse **TApplication** ist (für jedes Projekt wird automatisch ein **Application**-Objekt erstellt). Die Methode **CreateForm** von **TApplication** wird zweimal aufgerufen. Durch den ersten Aufruf **CreateForm** wird **Form1** erstellt, eine Instanz der in **Unit1** deklarierten Klasse **TForm1**. Durch den zweiten Aufruf von **CreateForm** wird **Form2** erstellt, eine Instanz der in **Unit2** deklarierten Klasse **TForm2**.

Unit1 hat folgenden Inhalt:

```

unit Unit1;

interface

uses SysUtils, Types, Classes, Graphics, Controls, Forms, Dialogs;

type

TForm1 = class(TForm)
  Button1: TButton;
  procedure Button1Click(Sender: TObject);
end;

var
  Form1: TForm1;

implementation

uses Unit2;

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
begin
  Form2.ShowModal;
end;

end.

```

Unit1 erstellt eine Klasse namens **TForm1** (abgeleitet von **TForm**) und eine Instanz dieser Klasse (**Form1**). **TForm1** enthält die Schaltfläche **Button1** (eine Instanz von **TButton**) und die Prozedur **Button1Click**, die aufgerufen wird, wenn der Benutzer auf **Button1** klickt. **Button1Click** verbirgt **Form1** und zeigt **Form2** an (durch den Aufruf von **Form2.ShowModal**).

Anmerkung: Im vorhergehenden Beispiel ist **Form2.ShowModal** von automatisch erstellten Formularen abhängig. Auch wenn sich solche Formulare gut für Beispielcode eignen, wird von ihrer Verwendung doch ausdrücklich abgeraten.

Das Objekt **Form2** ist in **Unit2** definiert:

```

unit Unit2;

interface

```

```
uses SysUtils, Types, Classes, Graphics, Controls, Forms, Dialogs;

type
TForm2 = class(TForm)
  Label1: TLabel;
  CancelButton: TButton;
  procedure CancelButtonClick(Sender: TObject);
end;

var
  Form2: TForm2;

implementation

uses Unit1;

{$R *.dfm}

procedure TForm2.CancelButtonClick(Sender: TObject);
begin
  Form2.Close;
end;

end.
```

Unit2 erstellt eine Klasse namens TForm2 und eine Instanz dieser Klasse (Form2). TForm2 enthält eine Schaltfläche (CancelButton, eine Instanz von TButton) und eine Beschriftung (Label1, eine Instanz von TLabel). Obwohl es aus dem Quelltext nicht ersichtlich ist, zeigt Label1 unseren Lieblingsgruß `Hello world!` an. Die Komponente ist in der Formulardatei von Form2, [Unit2.dfm](#), definiert.

TForm2 deklariert und definiert die Methode `CancelButtonClick`, die zur Laufzeit aufgerufen wird, wenn der Benutzer auf die Schaltfläche CancelButton klickt. Solche Prozeduren (wie auch `TForm1.Button1Click` in Unit1) nennt man *Ereignisbehandlungs Routinen*, weil sie auf Ereignisse reagieren, die zur Laufzeit der Anwendung eintreten. Sie werden durch die Formulardateien für Form1 und Form2 bestimmten Ereignissen zugewiesen.

Nach dem Starten der Anwendung greeting wird lediglich Form1 angezeigt, während Form2 nicht sichtbar ist (standardmäßig ist nur das erste in der Projektdatei erstellte Formular, das Hauptformular, zur Laufzeit sichtbar). Klickt der Benutzer auf die Schaltfläche in Form1, wird in Form2 die Meldung `Hello world!` angezeigt. Wenn der Benutzer auf CancelButton oder auf das Schließfeld in der Titelleiste klickt, wird Form2 geschlossen.

Siehe auch

Programme und Units ([siehe Seite 847](#))

Namespaces in Delphi ([siehe Seite 854](#))

4.1.1.2 Programme und Units

Dieses Kapitel enthält eine detaillierte Beschreibung der Programmorganisation in Delphi.

4.1.1.2.1 Programme und Units

Ein Delphi-Programm besteht aus einzelnen Quelltextmodulen, den so genannten Units. Die Units werden über ein spezielles Quelltextmodul verbunden, das einen **program**-, **library**- oder **package**-Kopf enthält. Jede Unit wird in einer eigenen Datei gespeichert und separat kompiliert. Die kompilierten Units werden dann zu einer Anwendung gelinkt. In RAD Studio sind hierarchisch strukturierte Namespaces hinzugekommen, die für noch mehr Flexibilität bei der Organisation von Units sorgen. Namespaces und Units bieten folgende Möglichkeiten:

- Aufteilung großer Programme in Module, die separat bearbeitet werden können.
- Erstellung von Bibliotheken, die von mehreren Programmen genutzt werden können.
- Weitergabe von Bibliotheken an andere Entwickler, ohne den Quelltext verfügbar zu machen.

Dieses Thema enthält Informationen zur Struktur einer Delphi-Anwendung und beschreibt den **program**-Kopf, die **unit**-Deklarationssyntax und die **uses**-Klausel. Auf Unterschiede zwischen der Win32- und der .NET-Plattform wird gesondert hingewiesen. Der Delphi-Compiler unterstützt .NET-Namespace auf der Win32-Plattform nicht. Der RAD Studio-Compiler unterstützt hierarchische .NET-Namespace. Einzelheiten finden Sie unter "Namespaces in Delphi".

Programmstruktur und -syntax

Eine vollständige, ausführbare Delphi-Anwendung besteht aus mehreren Unit-Modulen, die über ein Quelltextmodul, die **Projektdatei**, miteinander verbunden sind. Bei der herkömmlichen Pascal-Programmierung wird der gesamte Quelltext - einschließlich des Hauptprogramms - in **.pas**-Dateien gespeichert. CodeGear-Tools verwenden die Namenserweiterung **.dpr**, um das Quellmodul des Hauptprogramms zu kennzeichnen. Der weitere Quelltext befindet sich zum größten Teil in so genannten Unit-Dateien mit der Namenserweiterung **.pas**. Damit ein Projekt vollständig kompiliert werden kann, benötigt der Compiler neben der Projektquelle entweder eine Quelltextdatei oder eine kompilierte Unit-Datei für jede Unit.

Anmerkung: Es müssen keine Units in ein Projekt aufgenommen werden, alle Programme verwenden aber automatisch die Unit **System** sowie die Unit **SysInit**.

Die Quelltextdatei für eine ausführbare Delphi-Anwendung enthält folgende Elemente:

- Einen Programmkopf
- Eine **uses**-Klausel (optional)
- Einen Block mit Deklarationen und ausführbaren Anweisungen

Ein RAD Studio-Programm kann außerdem eine **namespaces**-Klausel enthalten, mit der zusätzliche Namespaces für die Suche nach generischen Units angegeben werden. Einzelheiten hierzu finden Sie unter "Namespaces in Delphi".

Der Compiler und somit auch die IDE setzt voraus, dass sich diese drei Elemente in einer einzelnen Projektdatei (**.dpr**) befinden.

Der Programmkopf

Der Programmkopf enthält den Namen des ausführbaren Programms. Er besteht aus dem reservierten Wort **program**, einem nachgestellten gültigen Bezeichner und einem abschließenden Strichpunkt. In Anwendungen, die mit CodeGear-Tools entwickelt werden, muss der Bezeichner dem Namen der Projektquelle entsprechen.

Das folgende Beispiel zeigt die Projektquelle für ein Programm namens **Editor**. Der Name der Projektdatei lautet demnach **Editor.dpr**.

```
program Editor;

uses Forms, REAbout, // Ein "Info"-Feld
      REMain;          // Hauptformular

{$R *.res}

begin
  Application.Title := 'Text Editor';
  Application.CreateForm(TMainForm, MainForm);
  Application.Run;
end.
```

Die erste Zeile enthält den Programmkopf mit dem reservierten Wort **program**. Über die **uses**-Klausel wird eine Abhängigkeit von drei zusätzlichen Units definiert: **Forms**, **REAbout** und **REMain**. Die Compiler-Direktive **{\$R** } linkt die Ressourcendatei des Projekts zum Programm. Der Block mit den Anweisungen zwischen den Schlüsselwörtern **begin** und **end** wird beim Starten des Programms ausgeführt. Die Projektdatei wird wie alle Delphi-Quelltextdateien mit einem Punkt (nicht mit einem Strichpunkt)

beendet.

Delphi-Projektdateien sind normalerweise kurz, da die Programmlogik üblicherweise in Unit-Dateien erstellt wird. Eine Delphi-Projektdatei enthält in der Regel nur den Code, der zum Starten des Hauptfensters der Anwendung und der Schleife für die Ereignisverarbeitung erforderlich ist. Projektdateien werden automatisch von der IDE generiert und verwaltet und nur in seltenen Fällen manuell bearbeitet.

In Standard-Pascal kann der Programmkopf hinter dem Programmnamen auch Parameter angeben:

```
program Calc(input, output);
```

Der Delphi-Compiler von CodeGear ignoriert diese Parameter.

In RAD Studio wird über den Programmkopf ein eigener Namespace, der so genannte Standardprojekt-Namespace, eingefügt. Entsprechendes gilt für die **library**- und **package**-Köpfe, wenn diese Projekttypen für die .NET-Plattform compiliert werden.

Die **uses**-Klausel

Die **uses**-Klausel gibt alle Units an, die in das Programm aufgenommen werden. Diese Units können eigene **uses**-Klauseln enthalten. Ausführliche Informationen zu **uses**-Klauseln in Unit-Quelldateien finden Sie weiter unten im Abschnitt *Unit-Referenzen und die uses-Klausel*.

Eine **uses**-Klausel besteht aus dem reservierten Wort **uses** und einer durch Kommas getrennten Liste der Units, von denen die Projektdatei direkt abhängig ist.

Der Block

Ein Block enthält eine einfache oder strukturierte Anweisung, die beim Starten des Programms ausgeführt wird. In den meisten **program**-Dateien besteht der Block aus einer zusammengesetzten Anweisung zwischen den reservierten Wörtern **begin** und **end**. Die einzelnen Anweisungen sind einfache Aufrufe von Methoden des **Application**-Objekts des Projekts. Die meisten Projekte verfügen über eine globale **Application**-Variable, die eine Instanz von **TApplication**, **TWebApplication** oder **TServiceApplication** enthält. Ein Block kann außerdem Deklarationen von Konstanten, Typen, Variablen, Prozeduren und Funktionen enthalten. Die Deklarationen müssen im Block vor den Anweisungen stehen.

Unit-Struktur und -Syntax

Eine Unit besteht aus Typen (einschließlich Klassen), Konstanten, Variablen und Routinen (Prozeduren und Funktionen). Jede Unit wird in einer separaten Quelldatei (**.pas**) definiert.

Eine Unit-Datei beginnt mit dem **unit**-Kopf, auf den das Schlüsselwort **interface** folgt. Nach dem Schlüsselwort **interface** wird in der **uses**-Klausel eine Liste mit Unit-Abhängigkeiten angegeben. Darauf folgen der **implementation**-Abschnitt und die optionalen **initialization**- und **finalization**-Abschnitte. Die Struktur einer Unit-Quelldatei sieht also folgendermaßen aus:

```
unit Unit1;

interface

uses // Liste der Unit-Abhängigkeiten...

implementation

uses // Liste der Unit-Abhängigkeiten...

// Implementierung der Klassenmethoden, -prozeduren und -funktionen...

initialization

// Unit-Initialisierungscode...

finalization

// Unit-Finalisierungscode...
```

end.

Eine Unit muss mit dem reservierten Wort **end** und einem Punkt abgeschlossen werden.

Der Unit-Kopf

Der Unit-Kopf gibt den Namen der Unit an. Er besteht aus dem reservierten Wort **unit**, einem nachgestellten gültigen Bezeichner und einem abschließenden Strichpunkt. In Anwendungen, die mit CodeGear-Tools entwickelt werden, muss der Bezeichner dem Namen der Unit-Datei entsprechen. Beispiel:

```
unit MainForm;
```

Dieser Unit-Kopf kann in einer Quelltextdatei namens [MainForm.pas](#) verwendet werden. Die Datei mit der kompilierten Unit trägt dann den Namen [MainForm.dcu](#) oder [MainForm.dcuil](#).

Unit-Namen müssen in einem Projekt eindeutig sein. Auch wenn die Unit-Dateien in unterschiedlichen Verzeichnissen gespeichert werden, dürfen in einem Programm keine Units mit identischen Namen verwendet werden.

Der interface-Abschnitt

Der **interface**-Abschnitt einer Unit beginnt mit dem reservierten Wort **interface**. Er endet mit dem Beginn des **implementation**-Abschnitts. Der **interface**-Abschnitt deklariert Konstanten, Typen, Variablen, Prozeduren und Funktionen, die für Clients verfügbar sind. Clients sind andere Units oder Programme, die auf Elemente aus dieser Unit zugreifen möchten. Solche Entitäten werden als *public* bezeichnet, da Code in anderen Units auf sie wie auf Entitäten zugreifen kann, die in der Unit selbst deklariert sind.

Die **interface**-Deklaration einer Prozedur oder Funktion enthält nur die Signatur der Routine. Die Signatur umfasst den Namen der Routine, Parameter und den Rückgabetyp (bei Funktionen). Der Block, der den ausführbaren Code für die Prozedur bzw. Funktion enthält, wird im **implementation**-Abschnitt definiert. Prozedur- und Funktionsdeklarationen im **interface**-Abschnitt entsprechen also **forward**-Deklarationen.

Die **interface**-Deklaration einer Klasse muss die Deklarationen aller Klassenelemente enthalten: Felder, Eigenschaften, Prozeduren und Funktionen.

Der **interface**-Abschnitt kann eine eigene **uses**-Klausel enthalten, die unmittelbar auf das Schlüsselwort **interface** folgen muss.

Der implementation-Abschnitt

Der **implementation**-Abschnitt einer Unit beginnt mit dem reservierten Wort **implementation** und endet mit dem Beginn des **initialization**-Abschnitts oder, wenn kein **initialization**-Abschnitt vorhanden ist, mit dem Ende der Unit. Der **implementation**-Abschnitt definiert Prozeduren und Funktionen, die im **interface**-Abschnitt deklariert wurden. Im **implementation**-Abschnitt können diese Prozeduren und Funktionen in beliebiger Reihenfolge definiert und aufgerufen werden. Sie brauchen in den Prozedur- und Funktionsköpfen keine Parameterlisten anzugeben, wenn diese im **implementation**-Abschnitt definiert werden. Geben Sie jedoch eine Parameterliste an, muss diese der Deklaration im **interface**-Abschnitt exakt entsprechen.

Außer den Definitionen der öffentlichen Prozeduren und Funktionen kann der **implementation**-Abschnitt Deklarationen von Konstanten, Typen (einschließlich Klassen), Variablen, Prozeduren und Funktionen enthalten, die für die Unit *privat* sind. Das heißt, dass andere Units nicht auf Entitäten zugreifen können, die im **implementation**-Abschnitt deklariert sind (im Gegensatz zu Entitäten im **interface**-Abschnitt).

Der **implementation**-Abschnitt kann eine eigene **uses**-Klausel enthalten, die unmittelbar auf das Schlüsselwort **implementation** folgen muss. Bezeichner, die in den im **implementation**-Abschnitt angegebenen Units deklariert sind, stehen nur innerhalb des **implementation**-Abschnitts zur Verfügung. Auf diese Bezeichner kann im **interface**-Abschnitt nicht Bezug genommen werden.

Der initialization-Abschnitt

Der **initialization**-Abschnitt ist optional. Er beginnt mit dem reservierten Wort **initialization** und endet mit dem Beginn des

finalization-Abschnitts oder - wenn kein **finalization**-Abschnitt vorhanden ist - mit dem Ende der Unit. Der **initialization**-Abschnitt enthält Anweisungen, die beim Programmstart in der angegebenen Reihenfolge ausgeführt werden. Arbeiten Sie beispielsweise mit definierten Datenstrukturen, können Sie diese im **initialization**-Abschnitt initialisieren.

Für Units in der **uses**-Liste von Interfaces werden die von einem Client verwendeten **initialization**-Abschnitte von Units in der Reihenfolge ausgeführt, in der Units in der **uses**-Klausel des Clients aufgeführt sind.

Der **finalization**-Abschnitt

Der **finalization**-Abschnitt ist optional und kann nur in Units verwendet werden, die auch einen **initialization**-Abschnitt enthalten. Dieser Abschnitt beginnt mit dem reservierten Wort **finalization** und endet mit dem Ende der Unit. Er enthält Anweisungen, die beim Beenden des Hauptprogramms ausgeführt werden (es sei denn, die *Halt*-Prozedur wird zur Terminierung des Programms verwendet). Im **finalization**-Abschnitt sollten Sie die Ressourcen freigeben, die im **initialization**-Abschnitt zugewiesen wurden.

finalization-Abschnitte werden in der umgekehrten Reihenfolge der **initialization**-Abschnitte ausgeführt. Initialisiert eine Anwendung beispielsweise die Units A, B und C in dieser Reihenfolge, werden die **finalization**-Abschnitte dieser Units in der Reihenfolge C, B und A ausgeführt.

Mit dem Beginn der Ausführung des **initialization**-Codes einer Unit ist sichergestellt, dass der zugehörige **finalization**-Abschnitt beim Beenden der Anwendung ausgeführt wird. Der **finalization**-Abschnitt muss deshalb auch unvollständig initialisierte Daten verarbeiten können, da der **initialization**-Code bei Auftreten eines Laufzeitfehlers möglicherweise nicht vollständig ausgeführt wird.

Anmerkung: Wenn der Code für die verwaltete .NET-Umgebung kompiliert wird, zeigen **initialization**- und **finalization**-Abschnitte ein anderes Verhalten. Weitere Informationen hierzu finden Sie im Kapitel "Speicherverwaltung".

Unit-Referenzen und die **uses**-Klausel

Eine **uses**-Klausel in einem Programm, einer Bibliothek oder einer Unit gibt die von diesem Modul verwendeten Units an. Die **uses**-Klausel kann an folgenden Positionen verwendet werden:

- Projektdatei eines Programms oder einer Bibliothek
- **interface**-Abschnitt einer Unit
- **implementation**-Abschnitt einer Unit

Die meisten Projektdateien enthalten, wie die **interface**-Abschnitte der meisten Units, eine **uses**-Klausel. Der **implementation**-Abschnitt einer Unit kann eine eigene **uses**-Klausel enthalten.

Die Units `System` und `SysInit` werden automatisch von jeder Anwendung verwendet und dürfen nicht in der **uses**-Klausel angegeben werden. (`System` implementiert Routinen für die Datei-E/A, String-Verarbeitung, Gleitkommaoperationen, dynamische Speicherzuweisung usw.). Andere Standard-Units (Bibliotheken) wie `SysUtils` müssen dagegen in der **uses**-Klausel angegeben werden. Normalerweise werden alle erforderlichen Units automatisch in die **uses**-Klausel platziert, wenn Sie in der IDE Units in ein Projekt einfügen oder daraus entfernen.

In **unit**-Deklarationen und **uses**-Klauseln müssen Unit-Namen mit den Dateinamen hinsichtlich der Groß-/Kleinschreibung übereinstimmen. In einem anderen Kontext (beispielsweise bei qualifizierten Bezeichnern) spielt bei Unit-Namen die Groß-/Kleinschreibung keine Rolle. Um Probleme mit Unit-Referenzen zu vermeiden, verweisen Sie auf die Unit-Quelldatei explizit:

```
uses MyUnit in "myunit.pas";
```

Enthält die Projektdatei eine solche explizite Referenz, können andere Quelldateien die Unit mit einer einfachen **uses**-Klausel referenzieren:

```
uses Myunit;
```

Die Syntax der **uses**-Klausel

Eine **uses**-Klausel besteht aus dem reservierten Wort **uses**, einem oder mehreren durch Kommas voneinander getrennten Unit-Namen und einem abschließenden Strichpunkt. Beispiele:

```
uses Forms, Main;

uses
  Forms,
  Main;

uses Windows, Messages, SysUtils, Strings, Classes, Unit2, MyUnit;
```

In der **uses**-Klausel eines Programms bzw. einer Bibliothek kann auf den Namen jeder Unit das reservierte Wort **in** mit dem Namen einer Quelltextdatei folgen. Der Name wird mit oder ohne Pfad in Hochkommas angegeben. Die Pfadangabe kann absolut oder relativ sein. Beispiele:

```
uses
  Windows, Messages, SysUtils,
  Strings in 'C:\Classes\Strings.pas', Classes;
```

Fügen Sie das Schlüsselwort **in** nach dem Unit-Namen ein, wenn Sie die Quelltextdatei einer Unit angeben müssen. Da in der IDE vorausgesetzt wird, dass die Unit-Namen den Namen der Quelltextdateien entsprechen, in denen sie gespeichert sind, ist die Angabe des Namens der Quelltextdatei normalerweise nicht erforderlich. Das reservierte Wort **in** wird nur benötigt, wenn die Position der Quelltextdatei aus folgenden Gründen nicht eindeutig ist:

- Die Quelltextdatei befindet sich in einem anderen Verzeichnis als die Projektdatei, und dieses Verzeichnis ist nicht im Suchpfad des Compilers aufgeführt.
- Mehrere Verzeichnisse im Suchpfad des Compilers enthalten Units mit identischen Namen.
- Sie compilieren in der Befehlszeile eine Konsolenanwendung und haben einer Unit einen Bezeichner zugeordnet, der nicht dem Namen der Quelltextdatei entspricht.

Der Compiler stellt mit der Konstruktion **in** ... auch fest, welche Units Teil eines Projekts sind. Nur Units, auf die in der **uses**-Klausel einer Projektdatei (**.dpr**) das reservierte Wort **in** und ein Dateiname folgt, werden als Teil des Projekts betrachtet. Alle anderen Units in der **uses**-Klausel werden zwar vom Projekt genutzt, gehören aber nicht zu ihm. Dieser Unterschied ist zwar für die Compilierung ohne Bedeutung, wird aber in IDE-Tools wie der Projektverwaltung sichtbar.

In der **uses**-Klausel einer Unit können Sie **in** nicht verwenden, um für den Compiler die Position einer Quelltextdatei anzugeben. Jede Unit muss sich im Suchpfad des Compilers befinden. Außerdem müssen die Namen der Units mit den Namen der Quelltextdateien identisch sein.

Mehrere und indirekte Unit-Referenzen

Die Reihenfolge der Units in der **uses**-Klausel bestimmt die Reihenfolge der Initialisierung dieser Units und wirkt sich auf die Suche des Compilers nach den Bezeichnern aus. Wenn zwei Units eine Variable, eine Konstante, einen Typ, eine Prozedur oder eine Funktion mit identischem Namen deklarieren, verwendet der Compiler die Deklaration der in der **uses**-Klausel zuletzt angegebenen Unit. Wollen Sie auf den Bezeichner in einer anderen Unit zugreifen, müssen Sie den vollständigen Bezeichnernamen angeben: Unitname.Bezeichner.

Eine **uses**-Klausel muss nur die Units enthalten, die direkt vom Programm bzw. von der Unit verwendet werden, in dem bzw. der die Klausel steht. Referenziert beispielsweise Unit A Konstanten, Typen, Variablen, Prozeduren oder Funktionen, die in Unit B deklariert sind, muss die Unit B explizit von Unit A verwendet werden. Referenziert B wiederum Bezeichner aus Unit C, ist Unit A indirekt von Unit C abhängig. In diesem Fall muss Unit C nicht in einer **uses**-Klausel in Unit A angegeben werden. Der Compiler benötigt jedoch Zugriff auf die Units B und C, während Unit A verarbeitet wird.

Das folgende Beispiel illustriert diese indirekte Abhängigkeit:

```
program Prog;
```

```

uses Unit2;
const a = b;
// ...

unit Unit2;
interface
uses Unit1;
const b = c;
// ...

unit Unit1;
interface
const c = 1;
// ...

```

Hier hängt das Programm `Prog` direkt von `Unit2` ab, die wiederum direkt von `Unit1` abhängig ist. `Prog` ist also indirekt von `Unit1` abhängig. Da `Unit1` nicht in der **uses**-Klausel von `Prog` angeben ist, sind die in `Unit1` deklarierten Bezeichner für `Prog` nicht verfügbar.

Damit ein Client-Modul compiliert werden kann, muss der Compiler Zugriff auf alle Units haben, von denen der Client direkt oder indirekt abhängt. Sofern der Quelltext dieser Units nicht geändert wurde, benötigt der Compiler nur die **.dcu**-Dateien (Win32) bzw. die **.dcu1**-Dateien (.NET), nicht jedoch die Quelltextdateien (**.pas**).

Werden im **interface**-Abschnitt einer Unit Änderungen vorgenommen, müssen die von dieser Unit abhängigen Units neu compiliert werden. Werden die Änderungen dagegen nur im **implementation**- oder einem anderen Abschnitt einer Unit vorgenommen, müssen die abhängigen Units nicht neu compiliert werden. Der Compiler überwacht diese Abhängigkeiten und nimmt Neucompilierungen nur vor, wenn dies erforderlich ist.

Zirkuläre Unit-Referenzen

Wenn sich Units direkt oder indirekt gegenseitig referenzieren, werden sie als gegenseitig voneinander abhängig bezeichnet. Gegenseitige Abhängigkeiten sind zulässig, wenn keine zirkulären Pfade auftreten, die eine **uses**-Klausel im **interface**-Abschnitt einer Unit mit der **uses**-Klausel im **interface**-Abschnitt einer anderen Unit verbinden. Ausgehend vom **interface**-Abschnitt einer Unit darf es also nicht möglich sein, über die Referenzen in den **uses**-Klauseln in **interface**-Abschnitten anderer Units wieder zu dieser Ausgangsklausel zu gelangen. Damit eine Gruppe gegenseitiger Abhängigkeiten gültig ist, muss der Pfad jeder zirkulären Referenz über die **uses**-Klausel mindestens eines **implementation**-Abschnitts führen.

Im einfachsten Fall mit zwei gegenseitig voneinander abhängigen Units bedeutet dies, dass die Units sich nicht gegenseitig in den **uses**-Klauseln der jeweiligen **interface**-Abschnitte referenzieren dürfen. Das folgende Beispiel führt also bei der Compilierung zu einem Fehler:

```

unit Unit1;
interface
uses Unit2;
// ...

unit Unit2;
interface
uses Unit1;
// ...

```

Eine legale gegenseitige Referenzierung ist jedoch möglich, indem eine der Referenzen in den **implementation**-Abschnitt verschoben wird:

```

unit Unit1;
interface
uses Unit2;
// ...

unit Unit2;
interface
//...

```

```
implementation
uses Unit1;
// ...
```

Um unzulässige zirkuläre Referenzen zu vermeiden, sollten Sie die Units möglichst in der **uses**-Klausel des **implementation**-Abschnitts angeben. Werden jedoch Bezeichner einer anderen Unit im **interface**-Abschnitt verwendet, muss die betreffende Unit in der **uses**-Klausel des **interface**-Abschnitts angegeben werden.

Siehe auch

Namespaces in Delphi (siehe Seite 854)

4.1.1.2.2 Namespaces in Delphi

In Delphi ist die Unit der grundlegende Container für Typen. Die Microsoft-Laufzeitumgebung (CLR, Common Language Runtime) führt jedoch eine weitere Organisationsebene ein: den Namespace. Im .NET Framework ist ein Namespace ein Container für Typen, in Delphi fungiert er als Container für Delphi-Units. Namespaces ermöglichen es, in Delphi auf Klassen des .NET Framework zuzugreifen und diese zu erweitern.

Im Gegensatz zu herkömmlichen Delphi-Units können Namespaces verschachtelt werden, um hierarchisch strukturierte Enthaltensbeziehungen zu definieren. Verschachtelte Namespaces dienen zur Organisation von Bezeichnern und Typen und sorgen gleichzeitig dafür, dass es zwischen Typen mit identischen Namen nicht zu Namenskonflikten kommt. Als Container für Delphi-Units können Namespaces auch verwendet werden, um zwischen gleichnamigen Units zu unterscheiden, die sich in verschiedenen Packages befinden.

So werden beispielsweise `MyClass` in `MyNameSpace` und `MyClass` in `YourNamespace` als unterschiedliche Klassen behandelt. Zur Laufzeit referenziert die CLR Klassen und Typen über ihren voll qualifizierten Namen. Dieser besteht aus dem Namen der Assemblierung gefolgt von dem Namespace, in dem sich der Typ befindet. Die CLR gibt kein Konzept für die Namespace-Hierarchie vor. Die Implementierung bleibt dem Programmierer überlassen.

Folgende Themen werden behandelt:

- Projekt-Standard-Namespace und Namespace-Deklaration
- Suchbereich für Namespaces
- Namespaces in Delphi-Units

Namespaces deklarieren

In RAD Studio führt eine Projektdatei (**program**, **library** oder **package**) implizit ihren eigenen Namespace ein: den *Projekt-Standard-Namespace*. Eine Unit kann Teil dieses Standard-Namespace sein oder so deklariert werden, dass sie ein Element eines anderen Namespace ist. Die Namespace-Zugehörigkeit einer Unit wird in jedem Fall im **unit**-Kopf festgelegt. Sehen Sie sich hierzu folgende explizite Namespace-Deklaration an:

```
unit MyCompany.MyWidgets.MyUnit;
```

Die Namespaces sind durch Punkte voneinander getrennt. Namespaces führen keine neuen Symbole für die Bezeichner zwischen den Punkten ein. Die Punkte sind Teil des Unit-Namens. Im obigen Beispiel ist `MyCompany.MyWidgets.MyUnit.pas` der Name der Quelldatei. Nach dem Compilieren lautet der Dateiname `MyCompany.MyWidgets.MyUnit.dcuil`.

Die Punkte bestimmen außerdem die Enthaltensbeziehung (Containment) der Namespaces, d. h. sie geben an, wie die Namespaces ineinander verschachtelt sind. In der obigen Deklaration ist `MyUnit` ein Element des Namespace `MyWidgets`, der wiederum im Namespace `MyCompany` enthalten ist. Die Enthaltensbeziehung dient lediglich Dokumentationszwecken.

Der Projekt-Standard-Namespace legt den Namespace für alle Units des Projekts fest. Sehen Sie sich hierzu die folgenden Deklarationen an:

```
Program MyCompany.Programs.MyProgram;
```

```
Library MyCompany.Libs.MyLibrary;
Package MyCompany.Packages.MyPackage;
```

Diese Anweisungen richten den Projekt-Standard-Namespace für ein Programm (**program**), eine Bibliothek (**library**) und ein Package (**package**) ein. Der Namespace ergibt sich, wenn der Bezeichner auf der rechten Seite (einschließlich Punkt) aus der Deklaration entfernt wird.

Eine Unit ohne explizite Namespace-Deklaration wird als *generische Unit* bezeichnet. Eine generische Unit ist automatisch ein Element des Projekt-Standard-Namespace. Ausgehend von der obigen **program**-Deklaration veranlasst die folgende Unit-Deklaration, dass der Compiler **MyUnit** als Element des Namespace **MyCompany.Packages** behandelt:

```
unit MyUnit;
```

Der Projekt-Standard-Namespace hat bei generischen Units keinen Einfluss auf den Namen der Delphi-Quelldatei. Ausgehend von obiger Deklaration würde dieser Name **MyUnit.pas** lauten. Der Compiler stellt dem **dcu1**-Dateinamen den Projekt-Standard-Namespace voran. Der Name der resultierenden **dcu1**-Datei würde demnach **MyCompany.Packages.MyUnit.dcu1** lauten.

Bei Namespace-Strings wird nicht zwischen Groß- und Kleinschreibung unterschieden. Namespaces, die sich nur hinsichtlich der Groß-/Kleinschreibung unterscheiden, werden vom Compiler als identisch betrachtet. Der Compiler behält die Schreibweise des Namespace jedoch bei und verwendet sie in Namen von Ausgabedateien, Fehlermeldungen und Unit-Angaben in Laufzeittypinformationen. Klassen- und Typnamen in Laufzeittypinformationen werden immer mit der kompletten Namespace-Spezifikation angegeben.

Suchbereich für Namespaces

Jede Unit muss alle anderen Units deklarieren, von denen sie abhängig ist. Wie auf der Win32-Plattform durchsucht auch der Compiler diese Units nach Bezeichnern. Für Units in expliziten Namespaces ist der Suchbereich bekannt. Für generische Units muss der Compiler jedoch einen Namespace-Suchbereich bestimmen.

Sehen Sie sich hierzu die folgenden **unit**- und **uses**-Deklarationen an:

```
unit MyCompany.ProjectX.ProgramY.MyUnit1;
uses MyCompany.Libs.Unit2, Unit3, Unit4;
```

MyUnit1 ist ein Element des Namespace **MyCompany.ProjectX.ProgramY** und von drei anderen Units abhängig: von **MyCompany.Libs.Unit2** sowie den generischen Units **Unit3** und **Unit4**. Bezeichnernamen in **Unit2** können vom Compiler problemlos aufgelöst werden, da die **uses**-Klausel den voll qualifizierten Unit-Namen enthält. Um Bezeichnernamen in **Unit3** und **Unit4** aufzulösen, muss der Compiler jedoch die Suchreihenfolge für die Namespaces bestimmen.

Suchreihenfolge für Namespaces

Suchpositionen können in drei Quellen definiert sein: In den Compiler-Optionen, im Standard-Projekt-Namespace und in dem Namespace der aktuellen Unit.

Der Compiler verwendet bei der Auflösung von Bezeichnernamen die folgende Suchreihenfolge:

1. Aktueller Unit-Namespace (falls vorhanden)
2. Projekt-Standard-Namespace (falls vorhanden)
3. Über Compiler-Optionen angegebene Namespaces

Beispiel für die Suche in Namespaces

Das folgende Beispielprojekt und die Unit-Dateien zeigen die Suchreihenfolge in Namespaces:

```
// Deklarationen für Projektdatei...
program MyCompany.ProjectX.ProgramY;
// Deklaration für Unit-Quelldatei...
unit MyCompany.ProjectX.ProgramY.MyUnit1;
```

Unter diesen Voraussetzungen durchsucht der Compiler die Namespaces in der folgenden Reihenfolge:

1. MyCompany.ProjectX.ProgramY
2. MyCompany.ProjectX
3. Über Compiler-Optionen angegebene Namespaces

Wenn die aktuelle Unit generisch ist (also keine explizite Namespace-Deklaration in der **unit**-Anweisung enthält), beginnt die Auflösung beim Projekt-Standard-Namespace.

Namespaces verwenden

In Delphi gelangt ein Modul über die **uses**-Klausel in den Kontext der aktuellen Unit. Das Modul muss in der **uses**-Klausel entweder mit dem voll qualifizierten Namen (einschließlich der kompletten Namespace-Spezifikation) oder über seinen generischen Namen referenziert werden. Im letzteren Fall kommt der Mechanismus zur Namespace-Auflösung zum Einsatz, um die Unit zu lokalisieren.

Voll qualifizierte Unit-Namen

Das folgende Beispiel zeigt eine **uses**-Klausel mit Namespaces:

```
unit MyCompany.Libs.MyUnit1
uses MyCompany.Libs.Unit2, // Voll qualifizierter Name
     UnitX;               // Generischer Name
```

Sobald sich ein Modul im Kontext befindet, können darin enthaltene Bezeichner über den nicht qualifizierten Namen oder über den voll qualifizierten Namen (falls mehrere Units einen Bezeichner gleichen Namens enthalten) referenziert werden. Die folgenden **writeln**-Anweisungen sind äquivalent:

```
uses MyCompany.Libs.Unit2;

begin
  writeln(MyCompany.Libs.Unit2.SomeString);
  writeln(SomeString);
end.
```

Ein voll qualifizierter Bezeichner muss die komplette Namespace-Spezifikation enthalten. Deshalb würde ein Bezug auf **SomeString**, der nur einen Teil des Namespace enthält, zu einem Fehler führen:

```
writeln(Unit2.SomeString);      // FEHLER!
writeln(Libs.Unit2.SomeString); // FEHLER!
writeln(MyCompany.Libs.Unit2.SomeString); // Richtig.
writeln(SomeString);           // Richtig.
```

Unzulässig ist es auch, nur einen Teil eines Namespace in der **uses**-Klausel anzugeben. Es gibt keinen Mechanismus, der alle Units und Symbole in einem Namespace importiert. Die folgende Anweisung importiert beispielsweise nicht alle Units und Symbole im Namespace **MyCompany**:

```
uses MyCompany; // FEHLER!
```

Diese Einschränkung gilt auch für die Anweisung **with-do**. Folgender Code führt zu einem Compiler-Fehler:

```
with MyCompany.Libs do // FEHLER!
```

Namespaces und .NET-Metadaten

Der Delphi für .NET-Compiler gibt nicht den gesamten Unit-Namen in der Punktnotation in die Assemblierung aus. Es wird nur der linke Teil - also alles, was links vom letzten Punkt im Namen steht - ausgegeben. Ein Beispiel:

```
unit MyCompany.MyClasses.MyUnit
```

Der Compiler gibt den Namespace **MyCompany.MyClasses** in die Metadaten der Assemblierung aus. Dadurch wird es anderen .NET-Sprachen erleichtert, Aufrufe in die Delphi-Assemblierungen durchzuführen.

Der Unterschied in den Namespace-Metadaten ist nur für externe Anwender der Assemblierung sichtbar. Der Delphi-Code in der Assemblierung behandelt den gesamten Namen in der Punktnotation weiterhin als den voll qualifizierten Namen.

Multi-Unit-Namespaces

Wenn die Unit-Deklarationen auf denselben Namespace verweisen, können mehrere Units zu demselben Namespace gehören. Beispielsweise könnten Sie zwei Dateien, unit1.pas und unit2.pas, mit den folgenden Unit-Deklarationen erstellen:

```
// in Datei 'unit1.pas'
unit MyCompany.ProjectX.ProgramY.Unit1
// in Datei 'unit2.pas'
unit MyCompany.ProjectX.ProgramY.Unit2
```

In diesem Beispiel enthält der Namespace `MyCompany.ProjectX.ProgramY` alle **interface**-Symbole aus `unit1.pas` und `unit2.pas`.

Symbolnamen in einem Namespace müssen für alle Units in dem Namespace eindeutig sein. Im obigen Beispiel wäre es ein Fehler für `Unit1` und `Unit2`, wenn beide ein globales Interface-Symbol namens `mySymbol` definierten.

Die einzelnen in einem Namespace gruppierten Units stehen dem Quelltext nur zur Verfügung, wenn die einzelnen Units explizit in der **uses**-Klausel der Datei enthalten sind. Das heißt: Wenn eine Quelldatei den Namespace verwendet, dann müssen die voll qualifizierte Bezeichnerausdrücke, die ein Symbol in einer Unit dieses Namespace referenzieren, den Namespace-Namen verwenden und nicht nur den Namen der Unit, die dieses Symbol definiert.

Eine **uses**-Klausel kann auf einen Namespace als auch auf einzelne Units in diesem Namespace verweisen. In diesem Fall kann ein voll qualifizierter Ausdruck, der auf ein Symbol in einer bestimmten, in der **uses**-Klausel aufgeführten Unit verweist, mit dem Unit-Namen oder dem voll qualifizierten Namespace-Namen (Namespace-Name und Unit-Name) für den Qualifizierer referenziert werden. Die beiden Referenzierungsarten sind identisch und verweisen auf dasselbe Symbol.

Anmerkung: Die explizite Verwendung einer Unit in der **uses**-Klausel funktioniert nur, wenn Sie Quelltext- oder `dcu`-Dateien compilieren.. Wenn die Namespace-Units in eine Assemblierung compiliert werden und die Assemblierung von dem Projekt anstatt von den einzelnen Units referenziert wird, dann schlägt der Quelltext, der eine Unit in dem Namespace explizit referenziert, fehl.

Siehe auch

Programme und Units (siehe Seite 847)

4.1.1.3 Grundlegende syntaktische Elemente

Dieser Abschnitt enthält eine Beschreibung der grundlegenden syntaktischen Elemente der Sprache Delphi.

4.1.1.3.1 Grundlegende syntaktische Elemente

In diesem Thema finden Sie einen Überblick über den Zeichensatz von Delphi und eine Beschreibung der Deklarationssyntax für folgende Komponenten:

- Bezeichner
- Zahlen
- Zeichen-Strings
- Labels
- Quelltextkommentare

Der Delphi-Zeichensatz

Die Delphi-Sprache verwendet den Unicode-Zeichensatz mit alphabetischen und alphanumerischen Unicode-Zeichen und

Unterstrichen. Die Sprache unterscheidet nicht zwischen Groß- und Kleinschreibung. Das Leerzeichen und die ASCII-Steuerzeichen (ASCII 0 bis 31 einschließlich ASCII 13 für Zeilenvorschub) werden als *Blanks* bezeichnet.

Der RAD Studio-Compiler akzeptiert eine Datei, die in UCS-2 oder UCS-4 codiert ist, wenn die Datei eine Byte-Reihenfolgenkennzeichnung (Byte Order Mark = BOM) enthält. Bei der Verwendung anderer Formate als UTF-8 kann jedoch die Compilergeschwindigkeit beeinträchtigt werden. Alle Zeichen in einer UCS-4-codierten Quelltextdatei müssen in UCS-2 ohne Ersatzpaare darstellbar sein. UCS-2-Codierungen mit Ersatzpaaren (einschließlich GB18030) werden nur akzeptiert, wenn die Compileroption `codepage` angegeben ist.

Kombinationen, die sich aus den grundlegenden syntaktischen Elementen (den so genannten *Token*) zusammensetzen, ergeben Ausdrücke, Deklarationen und Anweisungen. Eine *Anweisung* beschreibt eine algorithmische Aktion, die innerhalb eines Programms ausgeführt werden kann. Ein *Ausdruck* ist eine syntaktische Einheit, die in einer Anweisung enthalten ist und einen Wert beschreibt. Eine *Deklaration* definiert einen Bezeichner (z. B. den Namen einer Funktion oder Variablen), der in Ausdrücken und Anweisungen verwendet wird. Sie weist dem Bezeichner bei Bedarf auch Speicherplatz zu.

Der Delphi-Zeichensatz und grundlegende syntaktische Elemente

Im Prinzip ist ein Programm eine Folge von Token, die durch Trennzeichen voneinander getrennt sind. Token sind die kleinsten Texteinheiten in einem Programm. Ein Trennzeichen kann entweder ein Leerzeichen oder ein Kommentar sein. Prinzipiell ist die Trennung zweier Token durch ein Trennzeichen nicht in jedem Fall erforderlich. Das Quelltextfragment

```
Size := 20; Price := 10;
```

ist beispielsweise korrekt. Aufgrund gängiger Konventionen und zugunsten der besseren Lesbarkeit verdient aber die folgende Schreibweise den Vorzug:

```
Size := 20;
Price := 10;
```

Ein Token ist ein Symbol, ein Bezeichner, ein reserviertes Wort, eine Direktive, eine Ziffer, ein Label oder eine String-Konstante. Außer bei String-Konstanten kann ein Trennzeichen nicht Teil eines Token sein. Zwei aufeinanderfolgende Bezeichner, reservierte Wörter, Ziffern oder Labels müssen durch eines oder mehrere Trennzeichen voneinander getrennt werden.

Spezielle Symbole

Symbole sind nichtalphanumerische Zeichen bzw. Zeichenpaare, die eine feste Bedeutung haben. Als spezielle Symbole gelten folgende Einzelzeichen:

```
# $ & ' ( ) * + , - . / : ; < = > @ [ ] ^ { }
```

Die folgenden Zeichenpaare sind ebenfalls spezielle Symbole:

```
(* (. * ) .) .. // := <= >= <>
```

Die folgende Tabelle enthält äquivalente Symbole:

Spezielles Symbol	Entsprechendes Symbol
[(.
]	.)
{	(*
}	*)

Die öffnende eckige Klammer [ist gleichbedeutend mit dem Zeichenpaar, das aus einer öffnenden runden Klammer und einem Punkt besteht (.

Die schließende eckige Klammer] ist gleichbedeutend mit dem Zeichenpaar, das aus einem Punkt und einer schließenden runden Klammer besteht).

Die öffnende geschweifte Klammer { ist gleichbedeutend mit dem Zeichenpaar, das aus einer öffnenden runden Klammer und

einem Sternchen besteht (*).

Die schließende geschweifte Klammer } ist gleichbedeutend mit dem Zeichenpaar, das aus einem Sternchen und einer schließenden runden Klammer besteht *).

Anmerkung: Folgende Zeichen sind keine speziellen Symbole: %, ?, \, !, " (normales Anführungszeichen), _ (Unterstrich), | (Pipe) und ~ (Tilde).

Bezeichner

Bezeichner werden für Konstanten, Variablen, Felder, Typen, Eigenschaften, Prozeduren, Funktionen, Programme, Units, Bibliotheken und Packages verwendet. Obwohl ein Bezeichner beliebig lang sein kann, sind nur die ersten 255 Zeichen signifikant. Ein Bezeichner muss mit einem alphabetischen Zeichen oder einem Unterstrich (_) beginnen und darf keine Leerzeichen enthalten. Auf das erste Zeichen können alphanumerische Zeichen, Ziffern und Unterstriche folgen. Reservierte Wörter dürfen nicht als Bezeichner verwendet werden.

Anmerkung: Im .NET SDK wird empfohlen, Bezeichner nicht mit Unterstrichen zu beginnen, da dieses Namensmuster für Systemzwecke reserviert ist.

Da die Groß-/Kleinschreibung in Delphi nicht berücksichtigt wird, sind beispielsweise für den Bezeichner CalculateValue folgende Schreibweisen zulässig:

```
CalculateValue
calculateValue
calculatevalue
CALCULATEVALUE
```

Da Unit-Namen Dateinamen entsprechen, kann eine unterschiedliche Schreibweise zu Compilierungsfehlern führen. Weitere Informationen hierzu finden Sie unter *Unit-Referenzen und die uses-Klausel*.

Qualifizierte Bezeichner

Wenn Sie einen Bezeichner verwenden, der an mehreren Stellen deklariert wurde, muss dieser unter Umständen qualifiziert werden. Die Syntax für einen qualifizierten Bezeichner lautet:

Bezeichner1.*Bezeichner2*

Dabei qualifiziert *Bezeichner1* den *Bezeichner2*. Angenommen, zwei Units deklarieren eine Variable namens `CurrentValue`. Mit der folgenden Anweisung legen Sie fest, dass auf `CurrentValue` in `Unit2` zugegriffen werden soll:

```
Unit2.CurrentValue
```

Bezeichner können über mehrere Ebenen qualifiziert werden. Beispiel:

```
Form1.Button1.Click
```

Mit dieser Anweisung wird die Methode `Click` in `Button1` von `Form1` aufgerufen.

Wenn Sie einen Bezeichner nicht qualifizieren, wird seine Interpretation von den Regeln für den Gültigkeitsbereich festgelegt, die unter Blöcke und Gültigkeitsbereich (siehe Seite 862) beschrieben werden.

Erweiterte Bezeichner

Besonders bei der Programmierung in Delphi für .NET könnten Sie auf Bezeichner (z.B. Typen oder Methoden in Klassen) stoßen, die denselben Namen wie ein Delphi-Schlüsselwort haben. Eine Klasse könnte z.B. eine Methode mit dem Namen `begin` haben. Ein weiteres Beispiel ist die CLR-Klasse `Type` im Namespace `System`. `Type` ist ein Delphi-Schlüsselwort und darf nicht als Name für einen Bezeichner verwendet werden.

Wenn Sie den Bezeichner mit der vollständigen Namespace-Spezifikation qualifizieren, dann besteht kein Problem. Um

beispielsweise die Klasse Type einzusetzen, müssen Sie ihren voll qualifizierten Namen angeben:

```
var
  TMyType : System.Type; // Der voll qualifizierte Namespace wird verwendet, um
                        // die Doppeldeutigkeit mit dem Delphi-Schlüsselwort
auszuschließen.
```

Der Ampersandoperator (**&**) kann als eine kürzere Alternative zum Auflösen der Doppeldeutigkeit zwischen Bezeichnern und Delphi-Schlüsselwörtern verwendet werden. Wenn Sie auf eine Methode oder einen Typ mit demselben Namen wie ein Delphi-Schlüsselwort stoßen, können Sie die Namespace-Spezifikation weglassen. Sie müssen dann aber dem Bezeichnernamen ein Ampersand voranstellen. Im folgenden Code wird z.B. ein Ampersand zur Unterscheidung der CLR-Klasse Type von dem Delphi-Schlüsselwort **type** verwendet.

```
var
  TMyType : &Type; // Präfix '&' ist ok.
```

Reservierte Wörter

Die folgenden reservierten Wörter können weder neu definiert noch als Bezeichner verwendet werden.

Reservierte Wörter

and	else	inherited	packed	then
array	end	initialization	procedure	threadvar
as	except	inline	program	to
asm	exports	interface	property	try
begin	file	is	raise	type
case	final	label	record	unit
class	finalization	library	repeat	unsafe
const	finally	mod	resourcestring	until
constructor	for	nil	sealed	uses
destructor	function	not	set	var
dispinterface	goto	object	shl	while
div	if	of	shr	with
do	implementation	or	static	xor
downto	in	out	string	

Neben den oben aufgeführten Wörtern gelten in Objekttypdeklarationen auch **private**, **protected**, **public**, **published** und **automated** als reservierte Wörter. In allen anderen Fällen werden sie als Direktiven behandelt. Die Wörter **at** und **on** haben ebenfalls eine besondere Bedeutung.

Direktiven

Direktiven sind Wörter, die an bestimmten Stellen des Quelltextes besonders behandelt werden. Direktiven haben in Delphi spezielle Bedeutungen. Sie werden aber im Gegensatz zu reservierten Wörtern nur in Umgebungen verwendet, in denen benutzerdefinierte Bezeichner nicht auftreten können. Aus diesem Grund lassen sich Bezeichner definieren, die wie Direktiven lauten. (Dieses Vorgehen ist allerdings nicht empfehlenswert.)

Direktiven

absolute	dynamic	local	platform	requires
abstract	export	message	private	resident
assembler	external	name	protected	safecall
automated	far	near	public	stdcall
cdecl	forward	nodefault	published	stored
contains	implements	overload	read	varargs
default	index	override	readonly	virtual
deprecated	inline	package	register	write
dispid	library	pascal	reintroduce	writeonly

Zahlen

Integer- und Real-Konstanten lassen sich in dezimaler Notation als Folge von Ziffern ohne Kommas oder Leerzeichen darstellen. Das Vorzeichen wird durch den vorangestellten Operator + oder - angegeben. Werte sind per Vorgabe positiv (67258 ist also identisch mit +67258) und müssen im Bereich des größten vordefinierten Real- bzw. Integer-Typs liegen.

Zahlen mit Nachkommastellen oder Exponenten bezeichnen Real-Konstanten. Andere Dezimalzahlen sind Integer-Konstanten. Bei Real-Typen wird die wissenschaftliche Notation (E oder e gefolgt von einem Exponenten) als "mal 10 hoch" gelesen. So bedeutet 7E2 beispielsweise $7 * 10^2$, und 12.25e+6 oder 12.25e6 bedeutet $12.25 * 10^6$.

Das Dollarzeichen als Vorzeichen kennzeichnet eine hexadezimale Zahl, beispielsweise \$8F. Hexadezimalzahlen ohne vorausgehenden unären Operator - werden als positive Werte angesehen. Liegt ein Hexadezimalwert während einer Zuweisung außerhalb des Bereichs des empfangenden Typs, wird ein Fehler ausgelöst, ausgenommen beim **Integer** (32-Bit-Integer, wo eine Warnung ausgelöst wird). In diesem Fall werden Werte, die den positiven Bereich für Ganzzahlen überschreiten, als negative Zahlen angesehen. Dieses Verfahren entspricht der Zweierkomplement-Darstellung von Ganzzahlen.

Ausführliche Informationen über Real- und Integer-Typen finden Sie unter Datentypen (siehe Seite 887). Informationen über die Datentypen von Zahlen finden Sie unter Echte Konstanten (siehe Seite 926).

Labels

Ein Label ist ein Standard-Delphi-Bezeichner. Der Unterschied zu anderen Bezeichnern besteht darin, dass Labels mit einer Ziffer beginnen können. Numerische Labels können nicht mehr als zehn Ziffern enthalten, d. h. eine Zahl zwischen 0 und 9999999999.

Labels werden in **goto**-Anweisungen verwendet. Ausführliche Informationen über **goto**-Anweisungen und Labels finden Sie unter goto-Anweisungen (siehe Seite 862).

Zeichen-Strings

Ein Zeichen-String (auch String-Literal oder String-Konstante genannt) kann aus einem String in Anführungszeichen, einem Steuerzeichen-String oder einer Kombination aus beiden bestehen. Trennzeichen treten nur bei Strings in Anführungszeichen auf.

Ein String in Anführungszeichen setzt sich aus einer Folge von maximal 255 Zeichen des erweiterten ASCII-Zeichensatzes zusammen, muss in einer Zeile stehen und in halbe Anführungszeichen eingeschlossen sein. Ein String in Anführungszeichen, der zwischen den halben Anführungszeichen kein Zeichen enthält, ist ein Null-String. Zwei in einem String in Anführungszeichen unmittelbar aufeinanderfolgende halbe Anführungszeichen stehen für ein einzelnes Anführungszeichen. Beispiel:

```
'CodeGear' { CodeGear }
'You''ll see' { You'll see }
```

```
''''           { ' ' }
''             { Null-String }
''             { ein Leerzeichen }
```

Ein Steuerzeichen-String ist eine Folge von einem oder mehreren Steuerzeichen. Jedes dieser Steuerzeichen besteht aus einem #‐Symbol und einer vorzeichenlosen Integer‐Konstante zwischen 0 und 255 (dezimal oder hexadezimal), die das entsprechende ASCII‐Zeichen bezeichnet. Der Steuerzeichen‐String

```
#89#111#117
```

entspricht dem folgenden String in Anführungszeichen:

```
'You'
```

Sie können Strings in Anführungszeichen mit Steuerzeichen‐Strings kombinieren und damit längere Strings bilden. Beispielsweise wird mit dem String

```
'Zeile 1'#13#10'Zeile 2'
```

das Zeichen für einen Wagenrücklauf/Zeilenvorschub zwischen Zeile 1 und Zeile 2 eingefügt. Strings in Anführungszeichen lassen sich allerdings nicht auf diese Weise miteinander verbinden, weil ein Paar aufeinanderfolgender halber Anführungszeichen als ein einzelnes Zeichen interpretiert wird. (Für das Zusammenfügen von Strings in Anführungszeichen steht der Operator + zur Verfügung. Sie können die betreffenden Strings aber auch zu einem einzigen String in Anführungszeichen kombinieren.)

Die Länge eines Zeichen‐Strings entspricht der Anzahl der Zeichen im String. Zeichen‐Strings mit beliebiger Länge sind kompatibel zu allen String‐Typen und zum Typ **PChar**. Ein Zeichen‐String der Länge 1 ist kompatibel zu jedem Zeichentyp. Wenn die erweiterte Syntax ({\$X+}) aktiviert ist, ist ein nichtleerer String der Länge *n* kompatibel zu nullbasierten Arrays und gepackten Arrays mit *n* Zeichen. Weitere Informationen hierzu finden Sie unter Datentypen (siehe Seite 887).

Kommentare und Compiler‐Direktiven

Kommentare werden vom Compiler ignoriert. Dies gilt jedoch nicht für Kommentare, die als Trennzeichen (zur Trennung aufeinanderfolgender Token) oder als Compiler‐Direktiven fungieren.

Kommentare lassen sich auf verschiedene Arten kennzeichnen:

```
{ Text zwischen einer öffnenden und einer schließenden geschweiften Klammer. }
  (* Text zwischen einer öffnenden runden Klammer plus Stern und Stern plus rechter
schließender Klammer *)
    // Beliebiger Text zwischen einem doppelten Schrägstrich und dem Zeilenende.
```

Ähnliche Kommentare lassen sich nicht verschachteln. So funktioniert beispielsweise {{}}, der Kommentar (*{{}}*) jedoch nicht. Dies ist nützlich zum Auskommentieren von Abschnitten, die ebenfalls Kommentare enthalten.

Ein Kommentar, in dem unmittelbar nach { oder (* ein Dollarzeichen (\$) steht, ist eine Compiler‐Direktive. Beispiel:

```
{$WARNINGS OFF}
```

Diese Direktive weist den Compiler an, keine Warnmeldungen zu generieren.

Siehe auch

Ausdrücke (siehe Seite 877)

Deklarationen und Anweisungen (siehe Seite 862)

Unit‐Referenzen und die uses‐Klausel (siehe Seite 847)

4.1.1.3.2 Declarations and Statements

This topic describes the syntax of Delphi declarations and statements.

Aside from the **uses** clause (and reserved words like **implementation** that demarcate parts of a unit), a program consists

entirely of *declarations* and *statements*, which are organized into *blocks*.

This topic covers the following items:

- Declarations
- Simple statements such as assignment
- Structured statements such as conditional tests (e.g., `if-then`, and `case`), iteration (e.g., `for`, and `while`).

string;

The syntax and placement of a declaration depend on the kind of identifier you are defining. In general, declarations can occur only at the beginning of a block or at the beginning of the interface or **implementation** section of a unit (after the **uses** clause). Specific conventions for declaring variables, constants, types, functions, and so forth are explained in the documentation for those topics.

Hinting Directives

The 'hint' directives **platform**, **deprecated**, and **library** may be appended to any declaration. These directives will produce warnings at compile time. Hint directives can be applied to type declarations, variable declarations, class, interface and structure declarations, field declarations within classes or records, procedure, function and method declarations, and unit declarations.

When a hint directive appears in a unit declaration, it means that the hint applies to everything in the unit. For example, the Windows 3.1 style `OleAuto.pas` unit on Windows is completely deprecated. Any reference to that unit or any symbol in that unit will produce a deprecation message.

The **platform** hinting directive on a symbol or unit indicates that it may not exist or that the implementation may vary considerably on different platforms. The **library** hinting directive on a symbol or unit indicates that the code may not exist or the implementation may vary considerably on different library architectures.

The **platform** and **library** directives do not specify which platform or library. If your goal is writing platform-independent code, you do not need to know which platform a symbol is specific to; it is sufficient that the symbol be marked as specific to *some* platform to let you know it may cause problems for your goal of portability.

In the case of a procedure or function declaration, the hint directive should be separated from the rest of the declaration with a semicolon. Examples:

```
procedure SomeOldRoutine; stdcall deprecated;  
  
var  
  VersionNumber: Real library;  
  
type  
  AppError = class(Exception)  
  ...  
end platform;
```

When source code is compiled in the `{$HINTS ON}` `{$WARNINGS ON}` state, each reference to an identifier declared with one of these directives generates an appropriate hint or warning. Use **platform** to mark items that are specific to a particular operating environment (such as Windows or .NET), **deprecated** to indicate that an item is obsolete or supported only for backward compatibility, and **library** to flag dependencies on a particular library or component framework.

The RAD Studio compiler also recognizes the hinting directive **experimental**. You can use this directive to designate units which are in an unstable, development state. The compiler will emit a warning when it builds an application that uses the unit.

Declarations

The names of variables, constants, types, fields, properties, procedures, functions, programs, units, libraries, and packages are called *identifiers*. (Numeric constants like 26057 are not identifiers.) Identifiers must be declared before you can use them; the only exceptions are a few predefined types, routines, and constants that the compiler understands automatically, the variable `Result` when it occurs inside a function block, and the variable `Self` when it occurs inside a method implementation.

A declaration defines an identifier and, where appropriate, allocates memory for it. For example,

```
var Size: Extended;
declares a variable called Size that holds an Extended (real) value, while
function DoThis(X, Y: string): Integer;
```

declares a function called **DoThis** that takes two strings as arguments and returns an integer. Each declaration ends with a semicolon. When you declare several variables, constants, types, or labels at the same time, you need only write the appropriate reserved word once:

```
var
  Size: Extended;
  Quantity: Integer;
```

Statements

Statements define algorithmic actions within a program. Simple statements like assignments and procedure calls can combine to form loops, conditional statements, and other structured statements.

Multiple statements within a block, and in the initialization or finalization section of a unit, are separated by semicolons.

Simple Statements

A simple statement doesn't contain any other statements. Simple statements include assignments, calls to procedures and functions, and goto jumps.

Assignment Statements

An assignment statement has the form

variable := *expression*

where *variable* is any variable reference, including a variable, variable typecast, dereferenced pointer, or component of a structured variable. The *expression* is any assignment-compatible expression (within a function block, variable can be replaced with the name of the function being defined. See Procedures and functions (siehe Seite 931)). The := symbol is sometimes called the assignment operator.

An assignment statement replaces the current value of variable with the value of expression. For example,

```
I := 3;
```

assigns the value 3 to the variable *I*. The variable reference on the left side of the assignment can appear in the expression on the right. For example,

```
I := I + 1;
```

increments the value of *I*. Other assignment statements include

```
X := Y + Z;
Done := (I >= 1) and (I < 100);
Hue1 := [Blue, Succ(C)];
I := Sqr(J) - I * K;
Shortint(MyChar) := 122;
TByteRec(W).Hi := 0;
MyString[I] := 'A';
SomeArray[I + 1] := P^;
TMyObject.SomeProperty := True;
```

Procedure and Function Calls

A procedure call consists of the name of a procedure (with or without qualifiers), followed by a parameter list (if required). Examples include

```

PrintHeading;
Transpose(A, N, M);
Find(Smith, William);
Writeln('Hello world!');
DoSomething();
Unit1.SomeProcedure;
TMyObject.SomeMethod(X,Y);

```

With extended syntax enabled (`={$X+}`), function calls, like calls to procedures, can be treated as statements in their own right:

```
MyFunction(X);
```

When you use a function call in this way, its return value is discarded.

For more information about procedures and functions, see [Procedures and functions](#) (siehe Seite 931).

Goto Statements

A **goto** statement, which has the form

```
goto label
```

transfers program execution to the statement marked by the specified label. To mark a statement, you must first declare the label. Then precede the statement you want to mark with the label and a colon:

```
label: statement
```

Declare labels like this:

```
label label;
```

You can declare several labels at once:

```
label label1, ..., labeln;
```

A label can be any valid identifier or any numeral between 0 and 9999.

The label declaration, marked statement, and **goto** statement must belong to the same block. (See [Blocks and Scope](#), below.) Hence it is not possible to jump into or out of a procedure or function. Do not mark more than one statement in a block with the same label.

For example,

```

label StartHere;
...
StartHere: Beep;
goto StartHere;

```

creates an infinite loop that calls the `Beep` procedure repeatedly.

Additionally, it is not possible to jump into or out of a **try-finally** or **try-except** statement.

The **goto** statement is generally discouraged in structured programming. It is, however, sometimes used as a way of exiting from nested loops, as in the following example.

```

procedure FindFirstAnswer;
  var X, Y, Z, Count: Integer;
label FoundAnAnswer;
begin
  Count := SomeConstant;
  for X := 1 to Count do
    for Y := 1 to Count do
      for Z := 1 to Count do
        if ... { some condition holds on X, Y, and Z } then
          goto FoundAnAnswer;
        ...
        { Code to execute if no answer is found }
      Exit;

```

```
FoundAnAnswer:
  ... { Code to execute when an answer is found }
end;
```

Notice that we are using **goto** to jump out of a nested loop. Never jump into a loop or other structured statement, since this can have unpredictable effects.

Structured Statements

Structured statements are built from other statements. Use a structured statement when you want to execute other statements sequentially, conditionally, or repeatedly.

- A compound or **with** statement simply executes a sequence of constituent statements.
- A conditional statement that is an **if** or **case** statement executes at most one of its constituents, depending on specified criteria.
- Loop statements including **repeat**, **while**, and **for** loops execute a sequence of constituent statements repeatedly.
- A special group of statements including **raise**, **try...except**, and **try...finally** constructions create and handle exceptions. For information about exception generation and handling, see Exceptions (siehe Seite 977).

Compound Statements

A compound statement is a sequence of other (simple or structured) statements to be executed in the order in which they are written. The compound statement is bracketed by the reserved words **begin** and **end**, and its constituent statements are separated by semicolons. For example:

```
begin
  Z := X;
  X := Y;
  X := Y;
end;
```

The last semicolon before **end** is optional. So we could have written this as

```
begin
  Z := X;
  X := Y;
  Y := Z
end;
```

Compound statements are essential in contexts where Delphi syntax requires a single statement. In addition to program, function, and procedure blocks, they occur within other structured statements, such as conditionals or loops. For example:

```
begin
  I := SomeConstant;
  while I > 0 do
    begin
      .
      .
      I := I - 1;
    end;
end;
```

You can write a compound statement that contains only a single constituent statement; like parentheses in a complex term, **begin** and **end** sometimes serve to disambiguate and to improve readability. You can also use an empty compound statement to create a block that does nothing:

```
begin
end;
```

With Statements

A **with** statement is a shorthand for referencing the fields of a record or the fields, properties, and methods of an object. The syntax of a **with** statement is

1. **with** *obj* **do** *statement*, or
2. **with** *obj1*, ..., *objn* **do** *statement*

where *obj* is an expression yielding a reference to a record, object instance, class instance, interface or class type (metaclass) instance, and *statement* is any simple or structured statement. Within the *statement*, you can refer to fields, properties, and methods of *obj* using their identifiers alone, that is, without qualifiers.

For example, given the declarations

```
type
  TDate = record
    Day: Integer;
    Month: Integer;
    Year: Integer;
  end;
```

```
var
  OrderDate: TDate;
```

you could write the following **with** statement.

```
with OrderDate do
  if Month = 12 then
    begin
      Month := 1;
      Year := Year + 1;
    end
  else
    Month := Month + 1;
```

you could write the following **with** statement.

```
if OrderDate.Month = 12 then
  begin
    OrderDate.Month := 1;
    OrderDate.Year := OrderDate.Year + 1;
  end
else
  OrderDate.Month := OrderDate.Month + 1;
```

If the interpretation of *obj* involves indexing arrays or dereferencing pointers, these actions are performed once, before *statement* is executed. This makes **with** statements efficient as well as concise. It also means that assignments to a variable within *statement* cannot affect the interpretation of *obj* during the current execution of the **with** statement.

Each variable reference or method name in a **with** statement is interpreted, if possible, as a member of the specified object or record. If there is another variable or method of the same name that you want to access from the **with** statement, you need to prepend it with a qualifier, as in the following example.

```
with OrderDate do
  begin
    Year := Unit1.Year;
    ...
  end;
```

When multiple objects or records appear after **with**, the entire statement is treated like a series of nested **with** statements. Thus **with** *obj1*, *obj2*, ..., *objn* **do** *statement*

is equivalent to

```
with obj1 do
  with obj2 do
  ...
  with objn do
    // statement
```

In this case, each variable reference or method name in statement is interpreted, if possible, as a member of *objn*; otherwise it is interpreted, if possible, as a member of *obj1*; and so forth. The same rule applies to interpreting the *objs* themselves, so that, for instance, if *objn* is a member of both *obj1* and *obj2*, it is interpreted as *obj2.objn*.

If Statements

There are two forms of **if** statement: **if...then** and the **if...then...else**. The syntax of an **if...then** statement is

if *expression* **then** *statement*

where *expression* returns a **Boolean** value. If expression is **True**, then *statement* is executed; otherwise it is not. For example,

```
if J <> 0 then Result := I / J;
```

The syntax of an **if...then...else** statement is

if *expression* **then** *statement1* **else** *statement2*

where *expression* returns a **Boolean** value. If expression is **True**, then *statement1* is executed; otherwise *statement2* is executed. For example,

```
if J = 0 then
  Exit
else
  Result := I / J;
```

The **then** and **else** clauses contain one statement each, but it can be a structured statement. For example,

```
if J <> 0 then
  begin
    Result := I / J;
    Count := Count + 1;
  end
else if Count = Last then
  Done := True
else
  Exit;
```

Notice that there is never a semicolon between the **then** clause and the word **else**. You can place a semicolon after an entire **if** statement to separate it from the next statement in its block, but the **then** and **else** clauses require nothing more than a space or carriage return between them. Placing a semicolon immediately before **else** (in an **if** statement) is a common programming error.

A special difficulty arises in connection with nested **if** statements. The problem arises because some **if** statements have **else** clauses while others do not, but the syntax for the two kinds of statement is otherwise the same. In a series of nested conditionals where there are fewer **else** clauses than **if** statements, it may not seem clear which **else** clauses are bound to which **ifs**. Consider a statement of the form

if *expression1* **then** **if** *expression2* **then** *statement1* **else** *statement2*;

There would appear to be two ways to parse this:

if *expression1* **then** [**if** *expression2* **then** *statement1* **else** *statement2*];

if *expression1* **then** [**if** *expression2* **then** *statement1*] **else** *statement2*;

The compiler always parses in the first way. That is, in real code, the statement

```
if ... { expression1} then
  if ... {expression2} then
    ... {statement1}
```

```
else
  ... {statement2}
```

is equivalent to

```
if ... {expression1} then
  begin
    if ... {expression2} then
      ... {statement1}
    else
      ... {statement2}
  end;
```

The rule is that nested conditionals are parsed starting from the innermost conditional, with each **else** bound to the nearest available **if** on its left. To force the compiler to read our example in the second way, you would have to write it explicitly as

```
if ... {expression1} then
  begin
    if ... {expression2} then
      ... {statement1}
    end
  end
else
  ... {statement2};
```

Case Statements

The **case** statement may provide a readable alternative to deeply nested **if** conditionals. A **case** statement has the form

```
case selectorExpression of
  caseList1: statement1;
  ...
  caseListn: statementn;
end
```

where *selectorExpression* is any expression of an ordinal type smaller than 32 bits (string types and ordinals larger than 32 bits are invalid) and each *caseList* is one of the following:

- A numeral, declared constant, or other expression that the compiler can evaluate without executing your program. It must be of an ordinal type compatible with *selectorExpression*. Thus 7, **True**, 4 + 5 * 3, 'A', and `Integer('A')` can all be used as *caseLists*, but variables and most function calls cannot. (A few built-in functions like `Hi` and `Lo` can occur in a *caseList*. See Constant expressions (siehe Seite 926).)
- A subrange having the form *First..Last*, where *First* and *Last* both satisfy the criterion above and *First* is less than or equal to *Last*.
- A list having the form *item1*, ..., *itemn*, where each *item* satisfies one of the criteria above.

Each value represented by a *caseList* must be unique in the **case** statement; subranges and lists cannot overlap. A **case** statement can have a final **else** clause:

```
case selectorExpression of
  caseList1: statement1;
  ...
  caseListn: statementn;
  else
    statements;
end
```

where *statements* is a semicolon-delimited sequence of statements. When a **case** statement is executed, at most one of *statement1* ... *statementn* is executed. Whichever *caseList* has a value equal to that of *selectorExpression* determines the statement to be used. If none of the *caseLists* has the same value as *selectorExpression*, then the statements in the **else** clause (if there is one) are executed.

The **case** statement

```

case I of
  1..5: Caption := 'Low';
  6..9: Caption := 'High';
  0, 10..99: Caption := 'Out of range';
  else
    Caption := '';
end

```

is equivalent to the nested conditional

```

if I in [1..5] then
  Caption := 'Low';
else if I in [6..10] then
  Caption := 'High';
else if (I = 0) or (I in [10..99]) then
  Caption := 'Out of range'
else
  Caption := '';

```

Other examples of **case** statements

```

case MyColor of
  Red: X := 1;
  Green: X := 2;
  Blue: X = 3;
  Yellow, Orange, Black: X := 0;
end;

case Selection of
  Done: Form1.Close;
  Compute: calculateTotal(UnitCost, Quantity);
  else
    Beep;
end;

```

Control Loops

Loops allow you to execute a sequence of statements repeatedly, using a control condition or variable to determine when the execution stops. Delphi has three kinds of control loop: **repeat** statements, **while** statements, and **for** statements.

You can use the standard **Break** and **Continue** procedures to control the flow of a **repeat**, **while**, or **for** statement. **Break** terminates the statement in which it occurs, while **Continue** begins executing the next iteration of the sequence.

Repeat Statements

The syntax of a **repeat** statement is

repeat *statement1; ...; statmentn;until* *expression*

where *expression* returns a **Boolean** value. (The last semicolon before *until* is optional.) The **repeat** statement executes its sequence of constituent statements continually, testing *expression* after each iteration. When *expression* returns **True**, the **repeat** statement terminates. The sequence is always executed at least once because *expression* is not evaluated until after the first iteration.

Examples of **repeat** statements include

```

repeat
  K := I mod J;
  I := J;
  J := K;
until J = 0;

repeat
  Write('Enter a value (0..9): ');

```

```
Readln(I);
until (I >= 0) and (I <= 9);
```

While Statements

A **while** statement is similar to a **repeat** statement, except that the control condition is evaluated before the first execution of the statement sequence. Hence, if the condition is false, the statement sequence is never executed.

The syntax of a **while** statement is

while *expression* **do** *statement*

where *expression* returns a **Boolean** value and *statement* can be a compound statement. The **while** statement executes its constituent *statement* repeatedly, testing *expression* before each iteration. As long as *expression* returns **True**, execution continues.

Examples of **while** statements include

```
while Data[I] <> X do I := I + 1;

while I > 0 do
begin
  if Odd(I) then Z := Z * X;
  I := I div 2;
  X := Sqr(X);
end;

while not Eof(InputFile) do
begin
  Readln(InputFile, Line);
  Process(Line);
end;
```

For Statements

A **for** statement, unlike a **repeat** or **while** statement, requires you to specify explicitly the number of iterations you want the loop to go through. The syntax of a **for** statement is

for *counter* := *initialValue* **to** *finalValue* **do** *statement*

or

for *counter* := *initialValue* **downto** *finalValue* **do** *statement*

where

- *counter* is a local variable (declared in the block containing the **for** statement) of ordinal type, without any qualifiers.
- *initialValue* and *finalValue* are expressions that are assignment-compatible with *counter*.
- *statement* is a simple or structured statement that does not change the value of *counter*.

The **for** statement assigns the value of *initialValue* to *counter*, then executes *statement* repeatedly, incrementing or decrementing *counter* after each iteration. (The **for...to** syntax increments *counter*, while the **for...downto** syntax decrements it.) When *counter* returns the same value as *finalValue*, *statement* is executed once more and the **for** statement terminates. In other words, *statement* is executed once for every value in the range from *initialValue* to *finalValue*. If *initialValue* is equal to *finalValue*, *statement* is executed exactly once. If *initialValue* is greater than *finalValue* in a **for...to** statement, or less than *finalValue* in a **for...downto** statement, then *statement* is never executed. After the **for** statement terminates (provided this was not forced by a **Break** or an **Exit** procedure), the value of *counter* is undefined.

Warnung: The iteration variable *counter* cannot be modified within the loop. This includes assignment, and passing the variable to a **var** parameter of a procedure. Doing so results in a compile-time warning.

For purposes of controlling execution of the loop, the expressions *initialValue* and *finalValue* are evaluated only once, before the loop begins. Hence the **for...to** statement is almost, but not quite, equivalent to this **while** construction:

```
begin
  counter := initialValue;
  while counter <= finalValue do
  begin
    ... {statement};
    counter := Succ(counter);
  end;
end
```

The difference between this construction and the **for...to** statement is that the **while** loop reevaluates *finalValue* before each iteration. This can result in noticeably slower performance if *finalValue* is a complex expression, and it also means that changes to the value of *finalValue* within *statement* can affect execution of the loop.

Examples of **for** statements:

```
for I := 2 to 63 do
  if Data[I] > Max then
    Max := Data[I];

for I := ListBox1.Items.Count - 1 downto 0 do
  ListBox1.Items[I] := UpperCase(ListBox1.Items[I]);

for I := 1 to 10 do
  for J := 1 to 10 do
    begin
      X := 0;
      for K := 1 to 10 do
        X := X + Mat1[I,K] * Mat2[K,J];
      Mat[I,J] := X;
    end;

for C := Red to Blue do Check(C);
```

Iteration Over Containers Using For statements

Both Delphi for .NET and for Win32 support for-element-in-collection style iteration over containers. The following container iteration patterns are recognized by the compiler:

- for Element in ArrayExpr do Stmt;
- for Element in StringExpr do Stmt;
- for Element in SetExpr do Stmt;
- for Element in CollectionExpr do Stmt;
- for Element in Record do Stmt;

The type of the iteration variable *Element* must match the type held in the container. With each iteration of the loop, the iteration variable holds the current collection member. As with regular **for**-loops, the iteration variable must be declared within the same block as the **for** statement.

Warnung: The iteration variable cannot be modified within the loop. This includes assignment, and passing the variable to a **var** parameter of a procedure. Doing so results in a compile-time warning.

Array expressions can be single or multidimensional, fixed length, or dynamic arrays. The array is traversed in increasing order,

starting at the lowest array bound and ending at the array size minus one. The following code shows an example of traversing single, multi-dimensional, and dynamic arrays:

```

type
  TIntArray      = array[0..9] of Integer;
  TGenericIntArray = array of Integer;

var
  IArray1: array[0..9] of Integer  = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
  IArray2: array[1..10] of Integer = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
  IArray3: array[1..2] of TIntArray = ((11, 12, 13, 14, 15, 16, 17, 18, 19, 20),
                                       (21, 22, 23, 24, 25, 26, 27, 28, 29, 30));
  MultiDimTemp: TIntArray;
  IDynArray: TGenericIntArray;

  I: Integer;

begin
  for I in IArray1 do
    begin
      // Do something with I...
    end;

  // Indexing begins at lower array bound of 1.
  for I in IArray2 do
    begin
      // Do something with I...
    end;

  // Iterating a multi-dimensional array
  for MultiDimTemp in IArray3 do    // Indexing from 1..2
    for I in MultiDimTemp do        // Indexing from 0..9
      begin
        // Do something with I...
      end;

  // Iterating over a dynamic array
  IDynArray := IArray1;
  for I in IDynArray do
    begin
      // Do something with I...
    end;

```

The following example demonstrates iteration over string expressions:

```

var
  C: Char;
  S1, S2: String;
  Counter: Integer;

  OS1, OS2: ShortString;
  AC: AnsiChar;

begin
  S1 := 'Now is the time for all good men to come to the aid of their country.';
  S2 := '';

  for C in S1 do
    S2 := S2 + C;

  if S1 = S2 then
    WriteLn('SUCCESS #1');
  else
    WriteLn('FAIL #1');

```

```

OS1 := 'When in the course of human events it becomes necessary to dissolve...';
OS2 := '';

for AC in OS1 do
  OS2 := OS2 + AC;

if OS1 = OS2 then
  WriteLn('SUCCESS #2');
else
  WriteLn('FAIL #2');

end.

```

The following example demonstrates iteration over set expressions:

```

type
  TMyThing = (one, two, three);
  TMySet = set of TMyThing;
  TCharSet = set of Char;

var
  MySet: TMySet;
  MyThing: TMyThing;

  CharSet: TCharSet;
  {$IF DEFINED(CLR)}
  C: AnsiChar;
  {$ELSE}
  C: Char;
  {$IFEND}

begin
  MySet := [one, two, three];
  for MyThing in MySet do
    begin
      // Do something with MyThing...
    end;

  CharSet := [#0..#255];
  for C in CharSet do
    begin
      // Do something with C...
    end;

```

end.

To use the **for-in** loop construct on a class, the class must implement a prescribed collection pattern. A type that implements the collection pattern must have the following attributes:

- The class must contain a public instance method called `GetEnumerator()`. The `GetEnumerator()` method must return a class, interface, or record type.
- The class, interface, or record returned by `GetEnumerator()` must contain a public instance method called `MoveNext()`. The `MoveNext()` method must return a **Boolean**.
- The class, interface, or record returned by `GetEnumerator()` must contain a public instance, read-only property called `Current`. The type of the `Current` property must be the type contained in the collection.

If the enumerator type returned by `GetEnumerator()` implements the `IDisposable` interface, the compiler will call the `Dispose` method of the type when the loop terminates.

The following code demonstrates iterating over an enumerable container in Delphi.

```
type
```

```
TMYIntArray = array of Integer;

TMYEnumerator = class
  Values: TMYIntArray;
  Index: Integer;
public
  constructor Create;
  function GetCurrent: Integer;
  function MoveNext: Boolean;
  property Current: Integer read GetCurrent;
end;

TMYContainer = class
public
  function GetEnumerator: TMYEnumerator;
end;

constructor TMYEnumerator.Create;
begin
  inherited Create;
  Values := TMYIntArray.Create(100, 200, 300);
  Index := -1;
end;

function TMYEnumerator.MoveNext: Boolean;
begin
  if Index < High(Values) then
  begin
    Inc(Index);
    Result := True;
  end
  else
    Result := False;
end;

function TMYEnumerator.GetCurrent: Integer;
begin
  Result := Values[Index];
end;

function TMYContainer.GetEnumerator: TMYEnumerator;
begin
  Result := TMYEnumerator.Create;
end;

var
  MyContainer: TMYContainer;
  I: Integer;
  Counter: Integer;

begin
  MyContainer := TMYContainer.Create;
  Counter := 0;
  for I in MyContainer do
    Inc(Counter, I);
  Writeln('Counter = ', Counter);
end.
```

The following classes and their descendants support the **for-in** syntax:

- **TList**
- **TCollection**

- `TStrings`
- `TInterfaceList`
- `TComponent`
- `TMenuItem`
- `TCustomActionList`
- `TFields`
- `TListItems`
- `TTreeNodes`
- `TToolBar`

Blocks and Scope

Declarations and statements are organized into *blocks*, which define local namespaces (or *scopes*) for labels and identifiers. Blocks allow a single identifier, such as a variable name, to have different meanings in different parts of a program. Each block is part of the declaration of a program, function, or procedure; each program, function, or procedure declaration has one block.

Blocks

A block consists of a series of declarations followed by a compound statement. All declarations must occur together at the beginning of the block. So the form of a block is

```
{declarations}  
begin  
  {statements}  
end
```

The *declarations* section can include, in any order, declarations for variables, constants (including resource strings), types, procedures, functions, and labels. In a program block, the *declarations* section can also include one or more **exports** clauses (see [Libraries and packages](#) (siehe Seite 997)).

For example, in a function declaration like

```
function Uppercase(const S: string): string;  
var  
  Ch: Char;  
  L: Integer;  
  Source, Dest: PChar;  
begin  
  ...  
end;
```

the first line of the declaration is the function heading and all of the succeeding lines make up the block. `Ch`, `L`, `Source`, and `Dest` are local variables; their declarations apply only to the `Uppercase` function block and override, in this block only, any declarations of the same identifiers that may occur in the **program** block or in the **interface** or **implementation** section of a unit.

Scope

An identifier, such as a variable or function name, can be used only within the scope of its declaration. The location of a declaration determines its scope. An identifier declared within the declaration of a program, function, or procedure has a scope limited to the block in which it is declared. An identifier declared in the interface section of a unit has a scope that includes any other units or programs that use the unit where the declaration occurs. Identifiers with narrower scope, especially identifiers declared in functions and procedures, are sometimes called local, while identifiers with wider scope are called global.

The rules that determine identifier scope are summarized below.

If the identifier is declared in ...	its scope extends ...
the declaration section of a program, function, or procedure	from the point where it is declared to the end of the current block, including all blocks enclosed within that scope.
the interface section of a unit	from the point where it is declared to the end of the unit, and to any other unit or program that uses that unit. (See Programs and Units (siehe Seite 847).)
the implementation section of a unit, but not within the block of any function or procedure	from the point where it is declared to the end of the unit. The identifier is available to any function or procedure in the unit, including the initialization and finalization sections, if present.
the definition of a record type (that is, the identifier is the name of a field in the record)	from the point of its declaration to the end of the record-type definition. (See Records (siehe Seite 902).)
the definition of a class (that is, the identifier is the name of a data field property or method in the class)	from the point of its declaration to the end of the class-type definition, and also includes descendants of the class and the blocks of all methods in the class and its descendants. (See Classes and Objects (siehe Seite 948).)

Naming Conflicts

When one block encloses another, the former is called the outer block and the latter the inner block. If an identifier declared in an outer block is redeclared in an inner block, the inner declaration takes precedence over the outer one and determines the meaning of the identifier for the duration of the inner block. For example, if you declare a variable called `MaxValue` in the **interface** section of a unit, and then declare another variable with the same name in a function declaration within that unit, any unqualified occurrences of `MaxValue` in the function block are governed by the second, local declaration. Similarly, a function declared within another function creates a new, inner scope in which identifiers used by the outer function can be redeclared locally.

The use of multiple units further complicates the definition of scope. Each unit listed in a **uses** clause imposes a new scope that encloses the remaining units used and the **program** or **unit** containing the **uses** clause. The first unit in a **uses** clause represents the outermost scope and each succeeding unit represents a new scope inside the previous one. If two or more units declare the same identifier in their **interface** sections, an unqualified reference to the identifier selects the declaration in the innermost scope, that is, in the unit where the reference itself occurs, or, if that unit doesn't declare the identifier, in the last unit in the **uses** clause that does declare the identifier.

The `System` and `SysInit` units are used automatically by every program or unit. The declarations in `System`, along with the predefined types, routines, and constants that the compiler understands automatically, always have the outermost scope.

You can override these rules of scope and bypass an inner declaration by using a qualified identifier (see [Qualified Identifiers](#) (siehe Seite 857)) or a **with** statement (see [With Statements](#), above).

Siehe auch

[Fundamental Syntactic Elements](#) (siehe Seite 857)

[Expressions](#) (siehe Seite 877)

4.1.1.3.3 Ausdrücke

In diesem Thema werden die Syntaxregeln für Delphi-Ausdrücke beschrieben.

Folgende Bereiche werden behandelt:

- Gültige Delphi-Ausdrücke
- Operatoren

- Funktionsaufrufe
- Mengenkonstruktoren
- Indizes
- Typumwandlungen

Ausdrücke

Ein Ausdruck ist eine Konstruktion, die einen Wert zurückliefert. Die folgende Tabelle enthält Beispiele für Delphi-Ausdrücke:

X	Variable
@X	Adresse der Variable X
15	Integer-Konstante
InterestRate	Variable
Calc(X, Y)	Funktionsaufruf
X * Y	Produkt von X und Y
Z / (1 - Z)	Quotient von Z und (1 - Z)
X = 1.5	Boolescher Ausdruck
C in Range1	Boolescher Ausdruck
not Done	Negation eines Booleschen Ausdrucks
['a', 'b', 'c']	Menge
Char(48)	Wert einer Typumwandlung

Die einfachsten Ausdrücke sind Variablen und Konstanten (siehe Datentypen (siehe Seite 887)). Komplexere Ausdrücke werden mit Hilfe von Operatoren, Funktionsaufrufen, Mengenkonstruktoren, Indizes und Typumwandlungen aus einfachen Ausdrücken gebildet.

Operatoren

Operatoren verhalten sich wie vordefinierte Funktionen, die Bestandteil der Sprache Delphi sind. So setzt sich beispielsweise der Ausdruck (X + Y) aus den Variablen X und Y (den so genannten Operanden) und dem Operator + zusammen. Wenn X und Y den Typ **Integer** oder **Real** haben, liefert der Ausdruck (X + Y) die Summe der beiden Werte. Operatoren sind @, not, ^, *, /, div, mod, and, shl, shr, as, +, -, or, xor, =, >, <, <>, <=, >=, in und is.

Die Operatoren @, not und ^ sind unäre Operatoren und haben nur einen Operanden. Alle anderen Operatoren sind binär und haben zwei Operanden. Eine Ausnahme bilden die Operatoren + und -, die entweder unär oder binär sein können. Ein unärer Operator steht immer vor seinem Operanden (z. B. -B). Eine Ausnahme von dieser Regel bildet der Operator ^, der auf seinen Operanden folgt (z. B. P^). Binäre Operatoren stehen immer zwischen ihren Operanden (z. B. A = 7).

Das Verhalten einiger Operatoren hängt von dem Datentyp ab, der an sie übergeben wird. Beispielsweise führt der Operator not eine bitweise Negation eines Integer-Operanden und eine logische Negation eines Booleschen Operanden aus. Solche Operatoren sind deshalb auch mehreren Kategorien zugeordnet.

Mit Ausnahme von ^, is und in können alle Operatoren Operanden vom Typ Variant (siehe Seite 917) akzeptieren.

Die Erläuterungen in den folgenden Abschnitten gehen davon aus, dass Sie mit den Datentypen (siehe Seite 887) von Delphi vertraut sind.

Informationen zur Rangfolge der Operatoren in komplexen Ausdrücken finden Sie unter "Rangfolge von Operatoren" weiter unten.

Arithmetische Operatoren

Zu den arithmetischen Operatoren für Real- oder Integer-Operanden gehören die Operatoren **+**, **-**, *****, **/**, **div** und **mod**.

Binäre arithmetische Operatoren

Operator	Operation	Operandtyp	Ergebnistyp	Beispiel
+	Addition	Integer, Real	Integer, Real	<code>X + Y</code>
-	Subtraktion	Integer, Real	Integer, Real	<code>Ergebnis - 1</code>
*	Multiplikation	Integer, Real	Integer, Real	<code>P * InterestRate</code>
/	Gleitkommadivision	Integer, Real	Real	<code>X / 2</code>
div	Ganzzahlige Division	Integer	Integer	<code>Total div UnitSize</code>
mod	Rest	Integer	Integer	<code>Y mod 6</code>

Unäre arithmetische Operatoren

Operator	Operation	Operandtyp	Ergebnistyp	Beispiel
+	Positives Vorzeichen	Integer, Real	Integer, Real	<code>+7</code>
-	Negatives Vorzeichen	Integer, Real	Integer, Real	<code>-X</code>

Für arithmetische Operatoren gelten die folgenden Regeln:

- Der Wert von `x / y` ist vom Typ **Extended**, unabhängig vom Typ von `x` und `y`. Bei allen anderen Operatoren ist das Ergebnis vom Typ **Extended**, wenn mindestens ein Operand den Typ **Real** hat. Ist das nicht der Fall, ist das Ergebnis vom Typ **Int64**, wenn mindestens ein Operand den Typ **Int64** hat, ansonsten ist das Ergebnis vom Typ **Integer**. Wenn der Typ eines Operanden ein Unterbereich eines Integer-Typs ist, wird er wie ein Operand vom Typ **Integer** behandelt.
- Der Wert von `x div y` entspricht dem Wert von `x / y`, abgerundet in Richtung Null bis zum nächsten Integer-Wert.
- Der Operator **mod** liefert den Rest, der sich bei der Division seiner Operanden ergibt. Das bedeutet: $x \bmod y = x - (x \bmod y) * y$.
- Wenn `y` in einem Ausdruck der Form `x / y`, `x div y` oder `x mod y` den Wert Null hat, tritt ein Laufzeitfehler auf.

Boolesche Operatoren

Die Operanden der Booleschen Operatoren **not**, **and**, **or** und **xor** können einen beliebigen Booleschen Typ haben. Die Operatoren liefern einen Wert vom Typ **Boolean** zurück.

Boolesche Operatoren

Operator	Operation	Operandtyp	Ergebnistyp	Beispiel
not	Negation	Boolescher Ausdruck	Boolescher Ausdruck	<code>not (C in MySet)</code>
and	Konjunktion	Boolescher Ausdruck	Boolescher Ausdruck	<code>Done and (Total > 0)</code>
or	Disjunktion	Boolescher Ausdruck	Boolescher Ausdruck	<code>A or B</code>
xor	Exklusive Disjunktion	Boolescher Ausdruck	Boolescher Ausdruck	<code>A xor B</code>

Diese Operatoren folgen den Standardregeln der Booleschen Logik. Beispielsweise liefert ein Ausdruck der Form `x and y` nur dann den Wert **True**, wenn `x` und `y` den Wert **True** haben.

Vollständige Auswertung und Kurzschlussverfahren im Vergleich

Der Compiler unterstützt zwei Auswertungsmodi für die Operatoren **and** und **or**: die vollständige Auswertung und das Kurzschlussverfahren. Bei der vollständigen Auswertung werden alle Operanden eines **and**- oder **or**-Ausdrucks auch dann ausgewertet, wenn das Resultat des gesamten Ausdrucks bereits feststeht. Das Kurzschlussverfahren geht streng von links nach rechts vor und wird beendet, sobald das Ergebnis des gesamten Ausdrucks feststeht. Wenn beispielsweise der Ausdruck `A and B` im Kurzschlussverfahren ausgewertet wird und `A` den Wert **False** hat, wertet der Compiler `B` nicht mehr aus, da bereits feststeht, dass nach der Auswertung von `A` auch der gesamte Ausdruck den Wert **False** haben wird.

Das Kurzschlussverfahren ist normalerweise vorzuziehen, da es schneller ausgeführt wird und (in den meisten Fällen) einen geringeren Quelltextumfang erfordert. Die vollständige Auswertung ist von Nutzen, wenn ein Operand eine Funktion mit Nebeneffekten ist, die Änderungen in der Ausführung des Programms bewirken.

Das Kurzschlussverfahren ermöglicht auch Konstruktionen, die sonst zu unzulässigen Laufzeitoperationen führen würden. Die folgende Anweisung iteriert bis zum ersten Komma durch den String `S`:

```
while (I <= Length(S)) and (S[I] <> ',') do
  begin
    ...
    Inc(I);
  end;
```

Wenn `S` keine Kommas enthält, wird `I` bei der letzten Iteration auf einen Wert erhöht, der größer ist als `S`. Beim nachfolgenden Test der **while**-Bedingung wird bei einer vollständigen Auswertung versucht, `S[I]` zu lesen, was zu einem Laufzeitfehler führen kann. Beim Kurzschlussverfahren wird dagegen der zweite Teil der **while**-Bedingung (`S[I] <> ','`) nicht mehr ausgewertet, nachdem der erste Teil **False** ergibt.

Zur Steuerung des Auswertungsmodus dient die Compiler-Direktive `$/B`. Voreingestellt ist der Status `{$/B}` für das Kurzschlussverfahren. Um die vollständige Auswertung lokal zu aktivieren, fügen Sie die Direktive `{$/B+}` in den Quelltext ein. Sie können auch auf Projektebene in diesen Status wechseln, indem Sie im Dialogfeld Projektoptionen die Option Boolesche Ausdrücke vollständig aktivieren (alle Quell-Units müssen erneut compiliert werden).

Anmerkung: Wenn einer der Operanden vom Typ Variant ist, führt der Compiler immer eine vollständige Auswertung durch (auch im Status `{$/B}`).

Logische (bitweise) Operatoren

Die folgenden logischen Operatoren bearbeiten Integer-Operanden bitweise. Wenn z. B. der in `x` (binär) gespeicherte Wert 001101 und der in `y` gespeicherte Wert 100001 lautet, weist die Operation

```
Z := X or Y;
```

den Wert 101101 an `z` zu.

Logische (bitweise) Operatoren

Operator	Operation	Operandtyp	Ergebnistyp	Beispiel
not	Bitweise Negation	Integer	Integer	<code>not X</code>
and	Bitweises and	Integer	Integer	<code>X and Y</code>
or	Bitweises or	Integer	Integer	<code>X or Y</code>

xor	Bitweises xor	Integer	Integer	$x \text{ xor } y$
shl	Bitverschiebung nach links	Integer	Integer	$x \text{ shl } 2$
shr	Bitverschiebung nach rechts	Integer	Integer	$y \text{ shr } 1$

Für bitweise Operatoren gelten die folgenden Regeln:

- Das Ergebnis einer **not**-Operation hat denselben Typ wie der Operand.
- Wenn beide Operanden einer **and**-, **or**- oder **xor**-Operation Integer sind, hat das Ergebnis den vordefinierten Integer-Typ mit dem kleinsten Bereich, im dem alle für beide Typen möglichen Werte enthalten sind.
- Die Operationen $x \text{ shl } y$ und $x \text{ shr } y$ verschieben den Wert von x um y Bits nach links oder rechts (falls es sich bei x um einen vorzeichenlosen Integer handelt). Dies entspricht der Multiplikation oder Division von x durch 2^y . Das Ergebnis hat denselben Typ wie x . Wenn beispielsweise in N der Wert 01101 (dezimal 13) gespeichert ist, gibt $N \text{ sh } 1$ den Wert 11010 (dezimal 26) zurück. Beachten Sie, dass der Wert von y als Restwert der Größe von Typ x interpretiert wird. Wenn z. B. x ein Integer ist, wird $x \text{ shl } 40$ als $x \text{ shl } 8$ interpretiert, da ein Integer 32 Bit hat, und 40 minus 32 den Wert 8 ergibt.

String-Operatoren

Die relationalen Operatoren **=**, **<>**, **<**, **>**, **<=** und **>=** funktionieren auch mit String-Operanden (siehe "Relationale Operatoren"). Der Operator **+** verketten zwei Strings.

String-Operatoren

Operator	Operation	Operandtyp	Ergebnistyp	Beispiel
+	Verkettung	String, gepackter String, Char	String	<code>S + '.'</code>

Für die Verkettung von Strings gelten folgende Regeln:

- Die Operanden für **+** können Strings, gepackte Strings (gepackte Arrays vom Typ **Char**) oder Zeichen sein. Wenn jedoch ein Operand vom Typ **WideChar** ist, muss der andere Operand ein langer String (**AnsiString** oder **WideString**) sein.
- Das Ergebnis einer **+**-Operation ist mit allen String-Typen kompatibel. Wenn aber beide Operanden kurze Strings oder Zeichen sind und ihre gemeinsame Länge größer als 255 ist, wird das Ergebnis nach dem 255. Zeichen abgeschnitten.

Zeiger-Operatoren

Die relationalen Operatoren **<**, **>**, **<=** und **>=** können Operanden vom Typ **PChar** und **PWideChar** haben (siehe "Relationale Operatoren"). Bei den folgenden Operatoren können die Operanden auch Zeiger sein. Weitere Informationen über Zeiger finden Sie unter Zeiger und Zeigertypen (siehe Seite 911).

Zeichenzeiger-Operatoren

Operator	Operation	Operandtyp	Ergebnistyp	Beispiel
+	Zeigeraddition	Zeichenzeiger, Integer	Zeichenzeiger	<code>P + I</code>
-	Zeigersubtraktion	Zeichenzeiger, Integer	Zeichenzeiger, Integer	<code>P - Q</code>
^	Zeiger-Dereferenzierung	Zeigertypen	Basistyp von Zeiger	<code>P^</code>
=	Gleich	Zeigertypen	Boolescher Ausdruck	<code>P = Q</code>
<>	Ungleich	Zeigertypen	Boolescher Ausdruck	<code>P <> Q</code>

Der Operator **^** dereferenziert einen Zeiger. Sein Operand kann ein Zeiger auf einen beliebigen Typ mit Ausnahme des

generischen Typs **Pointer** sein, der vor der Dereferenzierung umgewandelt werden muss.

$P = Q$ ist nur dann **True**, wenn P und Q auf dieselbe Adresse zeigen. Andernfalls ergibt $P <> Q$ **True**.

Mit den Operatoren **+** und **-** kann der Offset eines Zeichenzeigers erhöht oder erniedrigt werden. Mit dem Operator **-** können Sie außerdem den Unterschied zwischen den Offsets zweier Zeichenzeiger berechnen. Für Zeiger-Operatoren gelten die folgenden Regeln:

- Wenn I ein Integer und P ein Zeichenzeiger ist, addiert $P + I$ den Wert I zu der von P angegebenen Adresse. Es wird also ein Zeiger auf die Adresse zurückgegeben, die I Zeichen hinter P liegt. (Der Ausdruck $I + P$ entspricht $P + I$.) $P - I$ subtrahiert I von der Adresse, die von P angegeben wird. Das Ergebnis ist ein Zeiger auf die Adresse, die I Zeichen vor P liegt. Dies gilt für **PChar**-Zeiger. Für **PWideChar**-Zeiger fügt $P + I$ `SizeOf(WideChar)` zu P hinzu.
- Wenn P und Q Zeichenzeiger sind, berechnet $P - Q$ die Differenz zwischen der von P angegebenen (höheren) und der von Q angegebenen (niedrigeren) Adresse. Es wird also ein Integer-Wert für die Anzahl der Zeichen zwischen P und Q zurückgegeben. Das Ergebnis von $P + Q$ ist nicht definiert.

Mengenoperatoren

Die folgenden Operatoren haben eine Menge als Operanden.

Mengenoperatoren

Operator	Operation	Operandtyp	Ergebnistyp	Beispiel
+	union	Menge	Menge	<code>Set1 + Set2</code>
-	Differenz	Menge	Menge	<code>S - T</code>
*	Schnittmenge	Menge	Menge	<code>S * T</code>
<=	Untermenge	Menge	Boolescher Ausdruck	<code>Q <= MySet</code>
>=	Obermenge	Menge	Boolescher Ausdruck	<code>S1 >= S2</code>
=	Gleich	Menge	Boolescher Ausdruck	<code>S2 = MySet</code>
<>	Ungleich	Menge	Boolescher Ausdruck	<code>MySet <> S1</code>
in	Element einer Menge	Ordinalwert, Menge	Boolescher Ausdruck	<code>A in Set1</code>

Für **+**, **-** und ***** gelten die folgenden Regeln:

- Der Ordinalwert O ist nur in $X + Y$ enthalten, wenn O in X oder Y (oder beiden) enthalten ist. O ist nur in $X - Y$ enthalten, wenn O in X , aber nicht in Y enthalten ist. O ist nur in $X * Y$ enthalten, wenn O sowohl in X als auch in Y enthalten ist.
- Das Ergebnis einer Operation mit **+**, **-** oder ***** ist vom Typ `set of A..B`, wobei A der kleinste und B der größte Ordinalwert in der Ergebnismenge ist.

Für **<=**, **>=**, **=**, **<>** und **in** gelten folgende Regeln.

- $X <= Y$ ist nur dann **True**, wenn jedes Element von X ein Element von Y ist. $Z >= W$ ist gleichbedeutend mit $W <= Z$. $U = V$ ist nur dann **True**, wenn U und V genau dieselben Elemente enthalten. Andernfalls gilt $U <> V$ ist **True**.
- Für einen Ordinalwert O und eine Menge S ist $O \in S$ nur dann **True**, wenn O ein Element von S ist.

Relationale Operatoren

Relationale Operatoren dienen dem Vergleich zweier Operanden. Die Operatoren **=**, **<>**, **<=** und **>=** lassen sich auch auf Mengen anwenden.

Relationale Operatoren

Operator	Operation	Operandtyp	Ergebnistyp	Beispiel
=	Gleich	Einfacher Typ, Klassen-, Klassenreferenz-, Interface-, String- und gepackter String-Typ	Boolescher Ausdruck	I = Max
<>	Ungleich	Einfacher Typ, Klassen-, Klassenreferenz-, Interface-, String- und gepackter String-Typ	Boolescher Ausdruck	X <> Y
<	kleiner als	Einfacher Typ, String-, gepackter String und PChar-Typ	Boolescher Ausdruck	X < Y
>	Größer als	Einfacher Typ, String-, gepackter String und PChar-Typ	Boolescher Ausdruck	Len > 0
<=	Kleiner oder gleich	Einfacher Typ, String-, gepackter String und PChar-Typ	Boolescher Ausdruck	Cnt <= I
>=	Größer oder gleich	Einfacher Typ, String-, gepackter String und PChar-Typ	Boolescher Ausdruck	I >= 1

Bei den meisten einfachen Typen ist der Vergleich unkompliziert. `I = J` ist beispielsweise nur dann **True**, wenn `I` und `J` denselben Wert haben. Andernfalls ist `I <> J` **True**. Für relationale Operatoren gelten die folgenden Regeln:

- Operanden müssen kompatible Typen haben, mit folgender Ausnahme: Reelle und Integer-Typen können miteinander verglichen werden.
- Strings werden gemäß den ordinalen Werten verglichen, die die Zeichen ausmachen, aus denen wiederum der String besteht. Zeichen-Typen werden als Strings der Länge 1 behandelt.
- Zwei gepackte Strings müssen beim Vergleich dieselbe Anzahl von Komponenten aufweisen. Wird ein gepackter String mit `n` Komponenten mit einem String verglichen, wird der gepackte String als String der Länge `n` behandelt.
- Die Operatoren `<`, `>`, `<=` und `>=` werden nur dann für den Vergleich von **PChar**-Operanden (sowie **PWideChar**) verwendet, wenn die beiden Zeiger auf Elemente in demselben Zeichen-Array zeigen.
- Die Operatoren `=` und `<>` können Operanden von Klassen- und Klassenreferenztypen haben. Mit Operanden eines Klassentyps werden `=` und `<>` entsprechend den Regeln für Zeiger ausgewertet: `C = D` ist nur dann **True**, wenn `C` und `D` auf dasselbe Instanzobjekt zeigen. Ansonsten ist `C <> D` **True**. Mit Operanden eines Klassenreferenztyps ist `C = D` nur dann **True**, wenn `C` und `D` dieselbe Klasse bezeichnen. Andernfalls ist `C <> D` **True**. Dies ist nicht mit Daten zu vergleichen, die in Klassen gespeichert sind. Weitere Informationen zu Klassen finden Sie unter Klassen und Objekte (siehe Seite 948).

Klassen-Operatoren

Die Operatoren **as** und **is** übernehmen Klassen und Instanzobjekte als Operanden; **as** kann auch auf Interfaces angewendet werden. Weitere Informationen finden Sie unter Klassen und Objekte (siehe Seite 948) und Objekt-Interfaces (siehe Seite 1006).

Die relationalen Operatoren `=` und `<>` können auch auf Klassen angewendet werden.

Der Operator @

Der Operator `@` liefert die Adresse einer Variablen, Funktion, Prozedur oder Methode, er erzeugt also einen Zeiger auf seinen Operanden. Weitere Informationen über Zeiger finden Sie unter Zeiger und Zeigertypen (siehe Seite 911). Für den Operator `@` gelten folgende Regeln:

- Ist `x` eine Variable, gibt `@x` die Adresse von `x` zurück. (Wenn `x` eine prozedurale Variable ist, gelten besondere Regeln; siehe Prozedurale Typen in Anweisungen und Ausdrücken (siehe Seite 914).) `@x` ist vom Typ **Pointer**, wenn die Standard-Compiler-Direktive `{$T}` aktiviert ist. Im Status `{$T+}` ist `@x` vom Typ `^T`, wobei `T` denselben Typ wie `x` hat (dieser Unterschied ist für die Zuweisungskompatibilität wichtig; siehe "Zuweisungskompatibilität").
- Wenn `F` eine Routine (eine Funktion oder Prozedur) ist, liefert `@F` den Eintrittspunkt von `F`. `@F` ist immer vom Typ **Pointer**.
- Wenn `@` auf eine in einer Klasse definierte Methode angewendet wird, muss der Methodenbezeichner mit dem

Klassennamen qualifiziert werden. Zum Beispiel:

```
@TMyClass.DoSomething
```

Diese Anweisung zeigt auf die Methode `DoSomething` von `TMyClass`. Weitere Informationen zu Klassen und Methoden finden Sie unter [Klassen und Objekte](#) (siehe Seite 948).

Anmerkung: Bei Verwendung des Operators `@` ist es nicht möglich, die Adresse einer Interface-Methode anzugeben, da die Adresse zur Compilierzeit nicht bekannt ist und zur Laufzeit nicht extrahiert werden kann.

4 Rangfolge von Operatoren

In komplexen Ausdrücken wird die Reihenfolge, in der Operationen ausgeführt werden, durch die Rangfolge der Operatoren festgelegt.

Rangfolge von Operatoren

Operatoren	Rangfolge
<code>@, not</code>	Erste (höchste)
<code>*, /, div, mod, and, shl, shr, as</code>	Zweite
<code>+, -, or, xor</code>	Dritte
<code>=, <>, <, >, <=, >=, in, is</code>	Vierte (niedrigste)

Ein Operator mit einer höheren Wertigkeit wird vor einem Operator mit einer niedrigeren Wertigkeit ausgewertet. Gleichrangige Operatoren werden von links nach rechts ausgewertet. Aus diesem Grund multipliziert der Ausdruck

`X + Y * Z`

zunächst `Y` mit `Z` und addiert dann `X` zum Ergebnis der Multiplikation. Die Operation `*` (Multiplikation) wird zuerst ausgeführt, weil dieser Operator höherwertig ist als der Operator `+`. Dagegen wird beim Ausdruck

`X - Y + Z`

zuerst `Y` von `X` subtrahiert und anschließend `Z` zum Ergebnis addiert. Die Operatoren `-` und `+` weisen dieselbe Wertigkeit auf. Aus diesem Grund wird die Operation auf der linken Seite zuerst ausgeführt.

Mit Hilfe von runden Klammern lassen sich die Rangfolgeregeln außer Kraft setzen. Ein Ausdruck in runden Klammern wird immer zuerst ausgewertet und danach wie ein einzelner Operand behandelt. Zum Beispiel:

`(X + Y) * Z`

Dieser Ausdruck multipliziert `Z` mit der Summe von `X` und `Y`.

Manchmal sind Klammern in Situationen erforderlich, die dies auf den ersten Blick nicht vermuten lassen. Betrachten Sie den folgenden Ausdruck:

`X = Y or X = Z`

Die beabsichtigte Interpretation lautet offensichtlich:

`(X = Y) or (X = Z)`

Wenn keine Klammern gesetzt werden, hält sich der Compiler an die Regeln für die Rangfolge der Operatoren und interpretiert den Ausdruck folgendermaßen:

`(X = (Y or X)) = Z`

Wenn `Z` nicht vom Typ **Boolean** ist, führt dies zu einem Compilierungsfehler.

Runde Klammern erleichtern zwar häufig das Schreiben und Lesen des Quelltextes, sind aber streng genommen überflüssig. Das erste der obigen Beispiele könnte auch folgendermaßen formuliert werden:

```
X + (Y * Z)
```

Hier sind die Klammern (für den Compiler) unnötig, erleichtern aber dem Programmierer und dem Leser die Interpretation, weil die Rangfolge der Operatoren nicht berücksichtigt werden muss.

Funktionsaufrufe

Funktionsaufrufe sind Ausdrücke, da sie einen Wert zurückliefern. Wenn Sie beispielsweise eine Funktion mit dem Namen Calc definiert haben, die zwei Integer-Argumente übernimmt und einen Integer-Wert zurückgibt, stellt der Funktionsaufruf Calc(24, 47) einen Integer-Ausdruck dar. Sind I und J Integer-Variablen, ist I + Calc(J, 8) ebenfalls ein Integer-Ausdruck. Hier einige Beispiele für Funktionsaufrufe:

```
Sum(A, 63)
Maximum(147, J)
Sin(X + Y)
Eof(F)
Volume(Radius, Height)
GetValue
TSomeObject.SomeMethod(I, J);
```

Weitere Informationen zu Funktionen finden Sie unter Prozeduren und Funktionen (siehe Seite 931).

Mengenkonstruktoren

Ein Mengenkonstruktur bezeichnet einen Wert eines Mengentyps. Zum Beispiel:

```
[5, 6, 7, 8]
```

Dieser Mengenkonstruktur bezeichnet eine Menge mit den Ziffern 5, 6, 7 und 8. Dieselbe Menge könnte auch mit dem Mengenkonstruktur

```
[5..8]
```

bezeichnet werden.

Die Syntax für einen Mengenkonstruktur lautet:

```
[Element1, ..., Elementn]
```

Hierbei kann *Element* ein Ausdruck sein, der einen Ordinalwert des Basistyps der Menge bezeichnet, oder ein Paar solcher Ausdrücke, die durch zwei Punkte (..) miteinander verbunden sind. Hat *Element* die Form x..y, bezeichnet es alle Ordinalwerte von x bis einschließlich y. Ist x jedoch größer als y, repräsentiert x..y keine Elemente, und [x..y] steht für eine leere Menge. Der Mengenkonstruktur [] bezeichnet die leere Menge, während [x] eine Menge repräsentiert, deren einziges Element der Wert x ist.

Hier einige Beispiele für Mengenkonstruktoren:

```
[red, green, MyColor]
[1, 5, 10..K mod 12, 23]
['A'..'Z', 'a'..'z', Chr(Digit + 48)]
```

Weitere Informationen zu Mengen finden Sie unter Mengen (siehe Seite 902).

Indizes

Strings, Arrays, Array-Eigenschaften und Zeiger auf Strings oder Arrays können indiziert werden. Beispielsweise liefert der Ausdruck Dateiname[3] für die String-Variable Dateiname den dritten Buchstaben in dem durch Dateiname bezeichneten String. Dagegen gibt Dateiname[I + 1] das Zeichen zurück, das unmittelbar auf das mit I indizierte Zeichen folgt. Weitere Informationen zu Strings finden Sie unter String-Typen (siehe Seite 896). Informationen zu Arrays und Array-Eigenschaften finden Sie unter Arrays (siehe Seite 902) und Array-Eigenschaften (siehe Seite 965).

Typumwandlungen

In bestimmten Situationen ist es erforderlich, den Typ eines Ausdrucks zu ändern. Durch eine Typumwandlung kann einem Ausdruck vorübergehend ein anderer Typ zugeordnet werden. Beispielsweise konvertiert die Anweisung `Integer('A')` den Buchstaben `A` in einen Integer.

Die Syntax für eine Typumwandlung lautet:

`Typebezeichner(Ausdruck)`

Handelt es sich bei dem Ausdruck um eine Variable, spricht man von einer Variablenumwandlung, andernfalls von einer Wertumwandlung. Obwohl die Syntax einer Wertumwandlung mit derjenigen einer Variablenumwandlung identisch ist, gelten für die beiden Umwandlungsarten unterschiedliche Regeln.

Wertumwandlungen

Bei einer Wertumwandlung müssen sowohl der Typebezeichner als auch der umzuwandelnde Ausdruck entweder ein ordinaler Typ oder ein Zeigertyp sein. Hier einige Beispiele für Wertumwandlungen:

```
Integer('A')
Char(48)
Boolean(0)
Color(2)
Longint(@Buffer)
```

Der resultierende Wert ergibt sich aus der Umwandlung des Ausdrucks in Klammern. Dabei wird der ursprüngliche Wert möglicherweise abgeschnitten oder erweitert, wenn die Größe des neuen Typs sich vom Typ des Ausdrucks unterscheidet. Das Vorzeichen des Ausdrucks bleibt aber in jedem Fall erhalten.

Die Anweisung

```
I := Integer('A');
```

weist der Variablen `I` den Wert von `Integer('A')` zu (also 65).

Auf eine Wertumwandlung dürfen keine Qualifizierer folgen. Außerdem sind Wertumwandlungen nur auf der rechten Seite einer Zuweisung erlaubt.

Variablenumwandlungen

Variablen können in jeden beliebigen Typ umgewandelt werden. Dabei muss allerdings die Größe gleich bleiben, und Integer-Typen dürfen nicht mit Real-Typen vermischt werden (verwenden Sie zur Umwandlung numerischer Typen Standardfunktionen wie `Int` und `Trunc`). Hier einige Beispiele für Variablenumwandlungen:

```
Char(I)
Boolean(Count)
TSomeDefinedType(MyVariable)
```

Variablenumwandlungen sind auf beiden Seiten einer Zuweisung erlaubt. Beispiel:

```
var MyChar: char;
  ...
  Shortint(MyChar) := 122;
```

Bei dieser Umwandlung wird `MyChar` das Zeichen `z` (ASCII 122) zugewiesen.

Variablen können in prozedurale Typen umgewandelt werden. Ausgehend von den Deklarationen

```
type Func = function(X: Integer): Integer;
var
  F: Func;
  P: Pointer;
  N : Integer;
```

sind z. B. folgende Zuweisungen möglich:

```

F := Func(P);      { Prozeduralen Wert in P an F zuweisen }
Func(P) := F;      { Prozeduralen Wert in F an P zuweisen }
@F := P;           { Zeigerwert in P an F zuweisen }
P := @F;           { Zeigerwert in F an P zuweisen }
N := F(N);         { Funktion über F aufrufen }
N := Func(P)(N);  { Funktion über P aufrufen }

```

Wie das folgende Beispiel zeigt, dürfen auf Variablenumwandlungen auch Qualifizierer folgen:

```

type
  TByteRec = record
    Lo, Hi: Byte;
  end;

  TWordRec = record
    Low, High: Word;
  end;

var
  B : Byte;
  W: Word;
  L : Longint;
  P: Pointer;

begin
  W := $1234;
  B := TByteRec(W).Lo;
  TByteRec(W).Hi := 0;
  L := $1234567;
  W := TWordRec(L).Low;
  B := TByteRec(TWordRec(L).Low).Hi;
  B := PByte(L)^;
end;

```

In diesem Beispiel wird mit `TByteRec` auf das niedrigst- und höchstwertige Byte eines Word zugegriffen. Über `TWordRec` erfolgt ein Zugriff auf das niedrigst- und höchstwertige Word eines Longint. Zu diesem Zweck könnten auch die vordefinierten Funktionen `Lo` und `Hi` verwendet werden. Die Variablenumwandlung bietet aber den Vorteil, dass sie auf der linken Seite einer Zuweisung stehen kann.

Weitere Informationen über die Typumwandlung von Zeigern finden Sie unter Zeiger und Zeigertypen (siehe Seite 911). Ausführliche Informationen zur Umwandlung von Klassen- und Interface-Typen finden Sie unter Der Operator `as` (siehe Seite 974) und Interface-Umwandlungen (siehe Seite 1013).

Siehe auch

Grundlegende syntaktische Elemente (siehe Seite 857)

Deklarationen und Anweisungen (siehe Seite 862)

4.1.1.4 Datentypen, Variablen und Konstanten

Dieser Abschnitt enthält eine Beschreibung der fundamentalen Datentypen der Sprache Delphi.

4.1.1.4.1 Datentypen, Variablen und Konstanten

Dieses Thema gibt einen Überblick über die Datentypen von Delphi.

Allgemeines zu Typen

Ein Typ ist im Wesentlichen ein Name für eine bestimmte Art von Daten. Wenn Sie eine Variable deklarieren, müssen Sie ihren Typ festlegen. Der Typ gibt an, welche Werte die Variable aufnehmen kann und welche Operationen mit ihr ausgeführt werden

können. Alle Ausdrücke und Funktionen geben Daten eines bestimmten Typs zurück. Den meisten Funktionen und Prozeduren müssen Parameter eines bestimmten Typs übergeben werden.

Delphi ist eine streng typisierte Sprache. Sie unterscheidet eine Vielzahl unterschiedlicher Datentypen, die nicht immer durch andere Typen ersetzbar sind. Diese Einschränkung ist normalerweise von Vorteil, da sie dem Compiler eine "intelligente" Datenbehandlung und eine gründliche Überprüfung des Quelltextes erlaubt, wodurch sich die Gefahr schwer diagnostizierbarer Laufzeitfehler verringert. Wenn Sie in bestimmten Fällen mehr Flexibilität benötigen, lässt sich diese strenge Typisierung mithilfe besonderer Techniken umgehen. Dazu gehören die Typumwandlung, Zeiger, Varianten, variante Teile in Record-Typen und die absolute Adressierung von Variablen.

Die Datentypen von Delphi können verschiedenen Kategorien zugeordnet werden.

- Einige Typen sind vordefiniert (oder integriert). Der Compiler erkennt diese Typen automatisch, so dass für sie keine Deklaration erforderlich ist. Die meisten der in dieser Sprachreferenz dokumentierten Typen sind vordefiniert. Andere Typen werden über eine Deklaration erzeugt. Dazu gehören benutzerdefinierte und in den Produktbibliotheken definierte Typen.
- Typen sind entweder fundamental oder generisch. Der Wertebereich und das Format fundamentaler Typen ist in allen Implementierungen von Delphi identisch, unabhängig von der zugrundeliegenden CPU und dem Betriebssystem. Der Wertebereich und das Format eines generischen Typs hingegen ist plattformspezifisch und von Implementierung zu Implementierung verschieden. Die meisten vordefinierten Typen sind fundamentale Typen, einige der Integer-, Zeichen-, String- und Zeigertypen sind jedoch generisch. Generell verdienen die generischen Typen den Vorzug, da sie eine optimale Ausführungsgeschwindigkeit und Portabilität gewährleisten. Wenn sich das Speicherformat eines generischen Typs jedoch in einer neuen Implementierung ändert, können Kompatibilitätsprobleme (beispielsweise beim Streamen von Daten in eine Datei als reine Binärdaten ohne Typ- und Versionsinformationen) auftreten.
- Typen lassen sich in einfache, String-, strukturierte, Zeiger-, prozedurale oder variante Typen einteilen. Auch Typbezeichner selbst gehören zu einem speziellen Typ, da sie als Parameter an bestimmte Funktionen (z. B. **High**, **Low** und **SizeOf**) übergeben werden können.

Die folgende Aufstellung zeigt die Struktur der Datentypen in Delphi.

Einfache Typen
Ordinaltypen
 Integer-Typen
 Zeichtypen
 Boolesche Typen
 Aufzählungstypen
 Teilbereichstypen
 Reelle Typen
String-Typen
Strukturierte Typen
 Mengentypen
 Array-Typen
 Record-Typen
 Dateitypen
 Klassentypen
 Klassenreferenztypen
 Interface-Typen
 Zeigertypen
 Prozedurale Typen
 Variante Typen
 Typbezeichner

Die Standardfunktion **SizeOf** kann alle Variablen und Typbezeichner verarbeiten. Sie liefert einen Integer-Wert zurück, der angibt, wie viele Bytes zum Speichern von Daten eines bestimmten Typs erforderlich sind. Beispielsweise liefert **SizeOf(Longint)** den Wert 4, weil eine **Longint**-Variable im Speicher vier Byte belegt.

Typdeklarationen werden in den folgenden Themen erläutert. Allgemeine Informationen zu Typdeklarationen finden Sie unter Typdeklaration (siehe Seite 923).

Siehe auch

Einfache Typen (siehe Seite 889)

- String-Typen (siehe Seite 896)
- Strukturierte Typen (siehe Seite 902)
- Zeiger und Zeigertypen (siehe Seite 911)
- Prozedurale Typen (siehe Seite 914)
- Variante Typen (siehe Seite 917)
- Kompatibilität und Identität von Typen (siehe Seite 921)
- Typdeklaration (siehe Seite 923)
- Variablen (siehe Seite 924)
- Deklarierte Konstanten (siehe Seite 926)

4.1.1.4.2 Einfache Typen

Einfache Typen, zu denen die ordinalen und reellen Typen gehören, definieren eine Menge von Werten mit eindeutiger Reihenfolge.

Die folgenden ordinalen Typen werden in diesem Thema behandelt:

- Integer-Typen
- Zeichentypen
- Boolesche Typen
- Aufzählungstypen
- Real-Typen (Gleitkomma)

Ordinal Typen

Zu den ordinalen Typen gehören Integer-, Zeichen-, Aufzählungs-, Teilbereichs- und Boolesche Typen. Ein ordinaler Typ definiert eine Menge von Werten mit eindeutiger Reihenfolge, in der jeder Wert mit Ausnahme des ersten einen eindeutigen Vorgänger und mit Ausnahme des letzten einen eindeutigen Nachfolger hat. Die Reihenfolge der Werte wird durch deren Ordinalposition festgelegt. In den meisten Fällen hat ein Wert mit der Ordinalposition n einen Vorgänger mit der Ordinalposition $n-1$ und einen Nachfolger mit der Ordinalposition $n+1$.

- Bei Integer-Typen ist die Ordinalposition mit dem Wert selbst identisch.
- Teilbereichstypen übernehmen die Ordinalposition von ihrem Basistyp.
- Bei allen anderen ordinalen Typen hat der erste Wert standardmäßig die Ordinalposition 0, der nächste die Ordinalposition 1 usw. In der Deklaration eines Aufzählungstyps kann diese Vorgabe überschrieben werden.

Einige vordefinierte Funktionen operieren mit ordinalen Werten und Typbezeichnern. Die wichtigsten dieser Funktionen sind in der folgenden Tabelle zusammengefasst.

Funktion	Parameter	Rückgabewert	Bemerkungen	
Ord	Ordinaler Ausdruck	Ordinalposition des Ausdruckswertes	Akzeptiert Int64-Argumente.	keine
Pred	Ordinaler Ausdruck	Vorgänger des Ausdruckswertes		
Succ	Ordinaler Ausdruck	Nachfolger des Ausdruckswertes		
High	Ordinaler Typbezeichner oder Variable mit ordinalem Typ	Höchster Wert des Typs	Verarbeitet auch kurze String-Typen und Arrays.	
Low	Ordinaler Typbezeichner oder Variable mit ordinalem Typ	Niedrigster Wert des Typs	Verarbeitet auch kurze String-Typen und Arrays.	

Beispielsweise liefert `High(Byte)` den Wert 255, weil 255 der höchste Wert des Typs **Byte** ist. `Succ(2)` liefert 3, weil 3 der Nachfolger von 2 ist.

Die Standardprozeduren `Inc` und `Dec` erhöhen bzw. erniedrigen den Wert der ordinalen Variable. Beispielsweise ist `Inc(I)` identisch mit `I := Succ(I)` oder mit `I := I + 1`, wenn `I` eine Integer-Variable ist.

Integer-Typen

Ein Integer-Typ repräsentiert eine Untermenge der ganzen Zahlen. Es gibt zwei generische Integer-Typen: **Integer** und **Cardinal**. Diese Typen sollten, wenn möglich, immer verwendet werden, da sie die optimale Ausführungsgeschwindigkeit für die zugrundeliegende CPU und das Betriebssystem gewährleisten. Die nachfolgende Tabelle enthält die Bereiche und Speicherformate der generischen Integer-Typen für den Delphi-Compiler.

Generische Integer-Typen

Typ	Bereich	Format	.NET-Entsprechung
Integer	-2147483648..2147483647	32 Bit, mit Vorzeichen	<code>Int32</code>
Cardinal	0..4294967295	32 Bit, ohne Vorzeichen	<code>UInt32</code>

Zu den fundamentalen Integer-Typen gehören **Shortint**, **Smallint**, **Longint**, **Int64**, **Byte**, **Word** und **Longword**.

Fundamentale Integer-Typen

Typ	Bereich	Format	.NET-Entsprechung
Shortint	-128..127	8 Bit, mit Vorzeichen	<code>SByte</code>
Smallint	-32768..32767	16 Bit, mit Vorzeichen	<code>Int16</code>
Longint	-2147483648..2147483647	32 Bit, mit Vorzeichen	<code>Int32</code>
Int64	-2^63..2^63-1	64 Bit, mit Vorzeichen	<code>Int64</code>
Byte	0..255	8 Bit, ohne Vorzeichen	<code>Byte</code>
Word	0..65535	16 Bit, ohne Vorzeichen	<code>UInt16</code>
Longword	0..4294967295	32 Bit, ohne Vorzeichen	<code>UInt32</code>
Largeuint	0..2^64-1	64 Bit, ohne Vorzeichen	<code>UInt64</code>

Generell gilt, dass arithmetische Operationen mit Integer-Werten einen Wert des Typs **Integer** zurückliefern, der in der aktuellen Implementierung mit dem 32-Bit-**Longint** identisch ist. Operationen liefern nur dann einen Wert vom Typ **Int64**, wenn sie für einen oder mehrere **Int64**-Operanden ausgeführt werden. Aus diesem Grund ergibt der folgende Quelltext kein korrektes Resultat:

```
var
  I: Integer;
  J : Int64;
  ...
  I := High(Integer);
  J := I + 1;
```

Um in dieser Umgebung einen Rückgabewert vom Typ **Int64** zu erhalten, muss für `I` eine Typumwandlung in **Int64** ausgeführt werden:

```
...
```

```
J := Int64(I) + 1;
```

Weitere Informationen hierzu finden Sie unter [Arithmetische Operatoren](#) (siehe Seite 877).

Anmerkung: Einige Standardroutinen mit Integer-Argumenten verkürzen **Int64**-Werte auf 32 Bit. Die Routinen **High**, **Low**, **Succ**, **Pred**, **Inc**, **Dec**, **IntToStr** und **IntToHex** unterstützen **Int64**-Argumente jedoch vollständig. Auch die Funktionen **Round**, **Trunc**, **StrToInt64** und **StrToInt64Def** geben **Int64**-Werte zurück. Einige wenige Routinen können keine **Int64**-Werte verarbeiten.

Wenn Sie den letzten Wert eines Integer-Typs erhöhen oder den ersten erniedrigen, erhalten Sie als Ergebnis den niedrigsten bzw. den höchsten Wert des Bereichs. Der Typ **Shortint** umfasst beispielsweise den Bereich 128..127. Deshalb hat nach Ausführung des Codes

```
var I: Shortint;
  ...
I := High(Shortint);
I := I + 1;
```

die Variable **I** den Wert 128. Wenn die Bereichsprüfung des Compilers eingeschaltet ist, führt dieser Code jedoch zu einem Laufzeitfehler.

Zeichentypen

Die fundamentalen Zeichentypen sind **AnsiChar** und **WideChar**. **AnsiChar**-Werte stellen Zeichen mit einer Breite von einem Byte (8 Bit) dar. Ihre Reihenfolge wird durch den Zeichensatz des Gebietsschemas festgelegt, wobei es sich auch um einen Multibyte-Zeichensatz handeln kann. **AnsiChar** war ursprünglich für den ANSI-Zeichensatz ausgelegt (daher auch der Name), wurde inzwischen aber so erweitert, dass der aktuelle Zeichensatz des Gebietsschemas berücksichtigt wird.

WideChar-Werte stellen Zeichen mit einer Länge von mehr als einem Byte dar. In aktuellen Implementierungen repräsentieren Werte des Typs **WideChar** Zeichen mit einer Breite von 16 Bit. Die Breite kann sich in zukünftigen Implementierungen erhöhen. Die Reihenfolge der Zeichen ist durch den Unicode-Zeichensatz definiert. Die ersten 256 Zeichen des Unicode-Zeichensatzes entsprechen den ANSI-Zeichen.

Der generische Zeichentyp ist **Char**. Er entspricht dem Typ **AnsiChar** in Win32 und dem Typ **Char** auf der .NET-Plattform. Die Implementierung des Typs **Char** kann sich in zukünftigen Versionen ändern. Wenn Sie Programme schreiben, in denen Zeichen unterschiedlicher Länge verarbeitet werden, sollten Sie deshalb statt hart codierter Konstanten die Standardfunktion **SizeOf** verwenden.

Anmerkung: Der Typ **WideChar** entspricht ebenfalls dem Typ **Char** von .NET.

Eine String-Konstante der Länge 1 (wie "A") kann einen Zeichenwert darstellen. Die vordefinierte Funktion **Chr** liefert den Zeichenwert aller Integer-Werte im Bereich von **AnsiChar** oder **WideChar**. So gibt beispielsweise **Chr(65)** den Buchstaben A zurück.

Wie Integer-Werte liefern auch Zeichenwerte den ersten bzw. den letzten Wert im Bereich, wenn der höchste Wert erhöht oder der niedrigste erniedrigt wird. Beispielsweise hat nach Ausführung des Codes

```
var
  Letter: Char;
  I: Integer;
begin
  Letter := High(Letter);
  for I := 1 to 66 do
    Inc(Letter);
end;
```

Letter den Wert A (ASCII 65).

Boolesche Typen

Es gibt vier vordefinierte Boolesche Typen: **Boolean**, **ByteBool**, **WordBool** und **LongBool**. In der Praxis wird in erster Linie der Typ **Boolean** verwendet. Die anderen Typen dienen der Kompatibilität zu anderen Sprachen und Betriebssystembibliotheken.

Eine **Boolean**-Variable belegt ebenso wie eine **ByteBool**-Variable ein Byte Speicherplatz. Eine **WordBool**-Variable belegt zwei (ein Word) und eine **LongBool**-Variable vier Bytes (zwei Word).

Boolesche Werte werden mit den vordefinierten Konstanten **True** und **False** dargestellt. Dabei gelten folgende Beziehungen:

Boolean	ByteBool, WordBool, LongBool
<i>False < True</i>	<i>False <> True</i>
<i>Ord(False) = 0</i>	<i>Ord(False) = 0</i>
<i>Ord(True) = 1</i>	<i>Ord(True) <> 0</i>
<i>Succ(False) = True</i>	<i>Succ(False) = True</i>
<i>Pred(True) = False</i>	<i>Pred(False) = True</i>

Ein Wert vom Typ **ByteBool**, **LongBool** oder **WordBool** hat den Wert **True**, wenn seine ordinale Position ungleich Null ist. Tritt ein derartiger Wert in einem Kontext auf, in dem ein Wert vom Typ **Boolean** erwartet wird, wandelt der Compiler automatisch einen Wert mit einer Ordinalposition ungleich Null in den Wert **True** um.

Die obigen Erläuterungen beziehen sich auf die Ordinalposition von Booleschen Werten, nicht jedoch auf die Werte selbst. In Delphi können Boolesche Ausdrücke nicht mit Integer- oder reellen Typen verglichen werden. Wenn beispielsweise X eine Integer-Variable ist, führt die Anweisung

```
if X then ...;
```

führt deshalb zu einem Compilierungsfehler. Die Umwandlung der Variable in einen Booleschen Typ ist nicht empfehlenswert. Verwenden Sie stattdessen eine der folgenden Alternativen:

```
if X <> 0 then ...; { Längeren Ausdruck verwenden, der einen Booleschen Wert liefert }
```

```
var OK: Boolean;
```

```
...
```

```
if X <> 0 then OK := True;
```

```
if OK then ...;
```

Aufzählungstypen

Aufzählungstypen definieren eine Menge von Werten mit eindeutiger Reihenfolge, indem einfach die einzelnen Bezeichner dieser Werte aneinander gereiht werden. Die Werte haben keine eigene Bedeutung. Die Syntax für die Deklaration eines Aufzählungstyps lautet:

```
type Typname = (Wert1, ..., Wertn)
```

Typname und *Wert* sind zulässige Bezeichner. Ein Beispiel:

```
type Suit = (Club, Diamond, Heart, Spade);
```

Hier wird ein Aufzählungstyp namens *Suit* mit den Werten Club, Diamond, Heart und Spade deklariert. *Ord(Club)* gibt in diesem Fall 0 zurück, *Ord(Diamond)* gibt 1 zurück usw.

Jeder Wert des Aufzählungstyps wird als Konstante des Typs *Typname* deklariert. Wenn die *Wert*-Bezeichner innerhalb desselben Gültigkeitsbereichs auch für einen anderen Zweck eingesetzt werden, können Namenskonflikte auftreten. Angenommen, Sie deklarieren folgenden Typ:

```
type TSound = (Click, Clack, Clock)
```

Click ist gleichzeitig der Name einer Methode, die für die Klasse `TControl` und für alle von ihr abgeleiteten Objekte in der VCL definiert ist. Wenn Sie eine Anwendung entwickeln und die folgende Ereignisbehandlungsroutine erstellen, tritt ein Compilierungsfehler auf:

```
procedure TForm1.DBGridEnter(Sender: TObject);
var Thing: TSound;
begin
  ...
  Thing := Click;
end;
```

Der Compiler interpretiert `Click` innerhalb des Gültigkeitsbereichs der Prozedur als Referenz auf die Methode `Click` von `TForm`. Sie können dieses Problem umgehen, indem Sie den Bezeichner qualifizieren. Wenn `TSound` beispielsweise in `MyUnit` deklariert ist, lautet die korrekte Anweisung

```
Thing := MyUnit.Click;
```

Die bessere Lösung besteht aber in der Verwendung von Konstantennamen, die nicht mit anderen Bezeichnern in Konflikt stehen. Beispiele:

```
type
  TSound = (tsClick, tsClack, tsClock);
  TMyColor = (mcRed, mcBlue, mcGreen, mcYellow, mcOrange);
  Answer = (ansYes, ansNo, ansMaybe)
```

Sie können die Konstruktion (*Wert1*, ..., *Wertn*) wie einen Typnamen direkt in einer Variablen Deklaration angeben:

```
var MyCard: (Club, Diamond, Heart, Spade);
```

Nach dieser Deklaration von `MyCard` ist es aber nicht mehr möglich, im selben Gültigkeitsbereich eine weitere Variable mit diesen Konstantenbezeichnern zu deklarieren. Die Anweisung:

```
var Card1: (Club, Diamond, Heart, Spade);
var Card2: (Club, Diamond, Heart, Spade);
```

führt deshalb zu einem Compilierungsfehler. Dagegen wird beim Ausdruck

```
var Card1, Card2: (Club, Diamond, Heart, Spade);
```

wird ebenso wie die folgenden Anweisungen fehlerfrei kompiliert:

```
type Suit = (Club, Diamond, Heart, Spade);
var
  Card1: Suit;
  Card2: Suit;
```

Aufzählungstypen mit expliziter Ordinalposition

Die Zählung der Ordinalposition von Aufzählungswerten beginnt standardmäßig bei 0 und entspricht dann der Reihenfolge, in der die Bezeichner in der Typdeklaration aufgeführt sind. Sie können diese Reihenfolge überschreiben und bestimmten oder allen Werten in der Deklaration eine explizite Ordinalposition zuweisen. Geben Sie dazu nach dem Bezeichner = *konstanterAusdruck* ein. Dabei ist *konstanterAusdruck* ein konstanter Ausdruck (siehe Seite 926), der zum einem Integer-Wert ausgewertet wird. Beispiel:

```
type Size = (Small = 5, Medium = 10, Large = Small + Medium);
```

Diese Deklaration definiert einen Aufzählungstyp namens `Size` mit den Werten `Small`, `Medium` und `Large`. `Ord(Small)` gibt in diesem Fall 5 zurück, `Ord(Medium)` 10 und `Ord(Large)` 15.

Im Grunde stellt ein Aufzählungstyp einen Teilbereich dar, dessen niedrigster und höchster Wert der niedrigsten und höchsten Ordinalposition der Konstanten in der Deklaration entsprechen. Der Typ `Size` im obigen Beispiel kann 11 Werte umfassen, deren Ordinalposition von 5 bis 15 reicht (der Typ `array[Size] of Char` repräsentiert also ein Array mit 11 Zeichen). Nur drei dieser Werte verfügen über einen Namen. Auf die anderen Werte kann über Typumwandlungen und Routinen wie `Pred`,

Succ, Inc und Dec zugegriffen werden. Im folgenden Beispiel werden der Variable x anonyme Wert im Bereich von Size zugewiesen.

```
var X: Size;
  X := Small;    // Ord(X) = 5
  Y := Size(6);  // Ord(X) = 6
  Inc(X);        // Ord(X) = 7
```

Die Ordinalposition eines Wertes ohne explizite Ordinalposition ist um 1 größer als die des vorhergehenden Wertes in der Liste. Wenn dem ersten Wert keine Ordinalposition zugewiesen wird, hat er die Position 0.

```
type SomeEnum = (e1, e2, e3 = 1);
```

SomeEnum hat nur zwei mögliche Werte: Ord(e1) gibt 0 zurück, Ord(e2) 1 und Ord(e3) ebenfalls 1. Da e2 und e3 dieselbe Ordinalposition haben, stellen sie denselben Wert dar.

Aufzählungskonstanten ohne festgelegten Wert besitzen Laufzeittypinformationen:

```
type SomeEnum = (e1, e2, e3);
```

Bei Aufzählungskonstanten mit einem festen Wert ist dies nicht der Fall:

```
type SomeEnum = (e1 = 1, e2 = 2, e3 = 3);
```

Teilbereichstypen

Ein Teilbereich ist eine Untermenge der Werte eines anderen ordinalen Typs (des so genannten Basistyps). Alle Konstruktionen der Form *Erster*..*Letzter*, in denen *Erster* und *Letzter* konstante Ausdrücke desselben ordinalen Typs sind und *Erster* kleiner ist als *Letzter*, bezeichnen einen Teilbereichstyp, der alle Werte von *Erster* bis *Letzter* enthält. Beispielsweise können Sie für den deklarierten Aufzählungstyp

```
type TColors = (Red, Blue, Green, Yellow, Orange, Purple, White, Black);
```

einen Teilbereichstyp der folgenden Form definieren:

```
type TMyColors = Green..White;
```

In diesem Fall umfasst TMyColors die Werte Green, Yellow, Orange, Purple und White.

Zur Definition von Teilbereichstypen können auch numerische Konstanten und Zeichen (String-Konstanten der Länge 1) verwendet werden:

```
type
  SomeNumbers = -128..127;
  Caps = 'A'..'Z';
```

Bei der Verwendung von numerischen oder Zeichenkonstanten ist der Basistyp der kleinste Integer- oder Zeichtyp, der den angegebenen Bereich enthält.

Die Konstruktion *LowerBound*..*UpperBound* funktioniert wie ein Typname, weshalb sie auch direkt in der Variablen Deklaration verwendet werden kann. Beispiel:

```
var SomeNum: 1..500;
```

Hier wird eine Integer-Variable deklariert, deren Wert im Bereich zwischen 1 und 500 liegt.

Die Ordinalposition der Werte eines Teilbereichs wird vom Basistyp bestimmt. (Wenn im ersten Beispiel Color eine Variable mit dem Wert Green ist, liefert Ord(Color) den Wert 2 zurück, unabhängig davon, ob Color vom Typ TColors oder TMyColors ist.) Dies gilt auch dann, wenn der Basistyp ein Integer- oder Zeichtyp ist. Eine Erhöhung oder Erniedrigung über die Grenzen eines Teilbereichs hinaus führt nur dazu, dass der Wert in den Basistyp umgewandelt wird. Während die Deklaration

```
type Percentile = 0..99;
```

```
var I: Percentile;
  ...
  I := 100;
```

einen Fehler ergibt, weist die Anweisung

```
...
  I := 99;
  Inc(I);
```

der Variablen `I` den Wert 100 zu (sofern die Bereichsprüfung des Compilers nicht eingeschaltet ist).

Die Verwendung von konstanten Ausdrücken in Teilbereichsdefinitionen bringt ein syntaktisches Problem mit sich. Wenn in einer Typdeklaration das erste bedeutungstragende Zeichen nach einem Gleichheitszeichen (=) eine öffnende Klammer ist, geht der Compiler davon aus, dass ein Aufzählungstyp definiert wird. Somit führen die Anweisungen

```
const
  X = 50;
  Y = 10;

type
  Scale = (X - Y) * 2 .. (X + Y) * 2;
```

zu einem Fehler. Sie können dieses Problem umgehen, indem Sie bei der Typdeklaration die führende Klammer vermeiden:

```
type
  Scale = 2 * (X - Y) .. (X + Y) * 2;
```

Reelle Typen

Ein reeller Typ definiert eine Menge von Zahlen, die in Gleitkommanotation dargestellt werden können. Die folgende Tabelle enthält die Bereiche und Speicherformate der fundamentalen reellen Typen in Win32.

Fundamentale reelle Typen in Win32

Typ	Bereich	Signifikante Stellen	Größe in Byte
Real48	-2,9 x 10^39 .. 1,7 x 10^38	11-12	6
Single	-1,5 x 10^45 .. 3,4 x 10^38	7-8	4
Double	-5,0 x 10^324 .. 1,7 x 10^308	15-16	8
Extended	-3,6 x 10^4951 .. 1,1 x 10^4932	10-20	10
Comp	-2^63+1 .. 2^63-1	10-20	8
Currency	-922337203685477.5808..922337203685477.5807	10-20	8

Die folgende Tabelle enthält eine Gegenüberstellung der fundamentalen reellen Typen und der entsprechenden Typen im .NET Framework

Entsprechungen für fundamentale reelle Typen in .NET

Typ	.NET-Entsprechung
Real48	Veraltet
Single	Single
Double	Double
Extended	Double
Comp	Veraltet

Currency	Wird unter Verwendung des .NET-Typs Decimal als Werttyp neu implementiert
-----------------	---

Der generische Typ **Real** entspricht in seiner aktuellen Implementierung dem Typ **Double** (und damit dem Typ Double in .NET).

Generische reelle Typen

Typ	Bereich	Signifikante Stellen	Größe in Byte
Real	-5,0 x 10 ³²⁴ .. 1,7 x 10 ³⁰⁸	15–16	8

Anmerkung: Der Typ **Real48** (6 Byte) hatte in früheren Object Pascal-Versionen den Namen **Real**. Wenn Sie Quelltext neu compilieren, der den alten Typ **Real** (6 Byte) in Delphi enthält, ändern Sie diesen Typ in **Real48**. Die Compiler-Direktive {\$REALCOMPATIBILITY ON} wandelt den Typ **Real** wieder in den alten 6-Byte-Typ um.

Die folgenden Erläuterungen beziehen sich auf die fundamentalen reellen Typen.

- **Real48** wird nur aus Gründen der Abwärtskompatibilität verwendet. Da das Speicherformat dieses Typs kein natives Format der Intel-Prozessorarchitektur ist, laufen die entsprechenden Operationen langsamer ab als mit anderen Gleitkommatypen. Der Typ **Real48** ist auf der .NET-Plattform veraltet.
- Der Typ **Extended** bietet eine höhere Genauigkeit, ist aber nicht so einfach portierbar wie die anderen reellen Typen. Verwenden Sie **Extended** mit Bedacht, wenn Sie Datendateien anlegen, die gemeinsam und plattformübergreifend genutzt werden sollen.
- Der Typ **Comp** (für "computational") ist ein natives Format der Intel-Prozessorarchitektur und stellt einen 64-Bit-Integer dar. Er ist dennoch als reeller Typ klassifiziert, weil sein Verhalten nicht dem eines ordinalen Typs entspricht. Ein **Comp**-Wert kann beispielsweise weder inkrementiert noch dekrementiert werden. **Comp** ist nur aus Gründen der Abwärtskompatibilität vorhanden. Eine höhere Ausführungsgeschwindigkeit erhalten Sie mit dem Typ **Int64**.
- Der Typ **Currency** ist ein Festkomma-Datentyp, der Rundungsfehler in finanzmathematischen Berechnungen minimiert. In Win32 wird er als skaliertes 64-Bit-Integer gespeichert, bei dem die vier niedrigst wertigen Stellen implizit vier Nachkommastellen repräsentieren. Bei einer Kombination mit anderen reellen Typen in Zuweisungen und Ausdrücken werden **Currency**-Werte automatisch mit 10000 multipliziert.

Siehe auch

Datentypen (siehe Seite 887)

String-Typen (siehe Seite 896)

Strukturierte Typen (siehe Seite 902)

Zeiger und Zeigertypen (siehe Seite 911)

Prozedurale Typen (siehe Seite 914)

Variante Typen (siehe Seite 917)

Kompatibilität und Identität von Typen (siehe Seite 921)

Typdeklaration (siehe Seite 923)

Variablen (siehe Seite 924)

Deklarierte Konstanten (siehe Seite 926)

4.1.1.4.3 String-Typen

Dieses Thema enthält eine Beschreibung der String-Datentypen, die in der Sprache Delphi zur Verfügung stehen. Folgende

Typen werden beschrieben:

- Kurze String-Typen
- Lange String-Typen
- Wide-String-Typen (Unicode)

Allgemeines zu String-Typen

Ein String-Typ stellt eine Folge von Zeichen dar. Delphi unterstützt die folgenden vordefinierten String-Typen:

String-Typen

Typ	Maximale Länge	Erforderlicher Speicherplatz	Verwendungszweck
ShortString	255 Zeichen	2 bis 256 Byte	Abwärtskompatibilität
AnsiString	$\sim 2^{31}$ Zeichen	4 Byte bis 2 GB	8-Bit-Zeichen (ANSI), DBCS ANSI, MBCS ANSI usw.
WideString	$\sim 2^{30}$ Zeichen	4 Byte bis 2 GB	Unicode-Zeichen, Mehrbenutzer-Server und mehrsprachige Anwendungen

In Win32 wird am häufigsten der Typ **AnsiString** (auch als langer String bezeichnet) verwendet, während auf der .NET-Plattform **WideString** der bevorzugte String-Typ ist.

String-Typen können in Zuweisungen und Ausdrücken miteinander kombiniert werden. Der Compiler führt die erforderlichen Umwandlungen automatisch durch. Strings, die als Referenz an eine Funktion oder Prozedur übergeben werden (z. B. als **var**- und **out**-Parameter), müssen jedoch den korrekten Typ aufweisen. Strings können explizit in einen anderen String-Typ umgewandelt werden.

Das reservierte Wort **string** funktioniert wie ein generischer Typbezeichner. Zum Beispiel:

```
var S: string;
```

Hier wird die Variable **S** für einen String erstellt. In Win32 interpretiert der Compiler **string** als **AnsiString** (wenn auf das reservierte Wort keine Zahl in eckigen Klammern folgt). Auf der .NET-Plattform entspricht der Typ **string** der Klasse **String**. Einzelbyte-Zeichenstrings sind in .NET zulässig, müssen aber explizit mit dem Typ **AnsiString** deklariert werden.

In Win32 können Sie die Direktive `{$H-}` verwenden, wenn **string** als **ShortString** interpretiert werden soll. Auf der .NET-Plattform ist die Direktive `{$H-}` veraltet.

Die Standardfunktion **Length** gibt die Anzahl der Zeichen in einem String zurück. Mit der Prozedur **SetLength** wird die Länge eines Strings festgelegt.

Der Vergleich von Strings wird durch die Wertigkeit der Zeichen an den entsprechenden Positionen definiert. Bei Vergleichen zwischen Strings von unterschiedlicher Länge wird jedes Zeichen im längeren String, dem kein Zeichen im kürzeren String entspricht, als "größer" angesehen. So ist beispielsweise "AB" größer als "A". Dies bedeutet, dass "AB" > "A" den Wert **True** hat. Strings mit der Länge Null enthalten die niedrigsten Werte.

Sie können eine String-Variable wie ein Array indizieren. Wenn **S** eine String-Variable und **i** ein Integer-Ausdruck ist, gibt **S[i]** das *i*-te Zeichen (genauer das *i*-te Byte in **S**) an. Bei einer **ShortString**- oder **AnsiString**-Variable ist **S[i]** vom Typ **AnsiChar**, bei einer **WideString**-Variable vom Typ **WideChar**. Für Einzelbyte-Gebietsschemas (Western) weist die Anweisung **MyString[2] := 'A';** dem zweiten Zeichen von **MyString** den Wert **A** zu. Im folgenden Quelltext wird **MyString** mit der Standardfunktion **AnsiUpperCase** in Großbuchstaben umgewandelt:

```
var I: Integer;
begin
  I := Length(MyString);
  while I > 0 do
    begin
```

```

MyString[I] := AnsiUpperCase(MyString[I]);
I := I -1;
end;
end;

```

Wenn Sie Strings auf diese Weise indizieren, müssen Sie darauf achten, dass Sie nicht über das Ende des Strings hinausschreiben, da dies zu einer Zugriffsverletzung führen würde. Außerdem sollten Sie Indizes für lange Strings nicht als **var**-Parameter übergeben, da dies ineffizienten Code ergibt.

Sie können einer String-Variablen den Wert einer String-Konstanten oder eines anderen Ausdrucks zuweisen, der einen String zurückliefert. Die Länge des Strings ändert sich bei der Zuweisung dynamisch. Beispiele:

```

MyString := 'Hello world!';
MyString := 'Hello' + 'world';
MyString := MyString + '!';
MyString := ' '; { Leerzeichen }
MyString := ''; { Leer-String }

```

Kurze String-Typen

Ein **ShortString** hat eine Länge von 0 bis 255 Zeichen. Obwohl sich seine Länge dynamisch ändern kann, beträgt die statische Speicherplatzzuweisung immer 256 Bytes. Im ersten Byte wird die Länge des Strings gespeichert, die restlichen 255 Byte stehen für die Zeichen zur Verfügung. Wenn **S** eine **ShortString**-Variable ist, gibt **Ord(S[0])** die Länge von **S** zurück (dasselbe Ergebnis erzielen Sie mit **Length(S)**). Durch Zuweisung eines Wertes an **S[0]** können Sie (wie durch einen Aufruf von **SetLength**) die Länge von **S** ändern. **ShortString** wird nur aus Gründen der Abwärtskompatibilität mitgeführt.

Delphi unterstützt kurze String-Typen (Untertypen von **ShortString**), deren maximale Länge zwischen 0 und 255 Zeichen liegen kann. Diese Typen werden mit einer Zahl in eckigen Klammern dargestellt, die auf das reservierte Wort **string** folgt. Zum Beispiel:

```
var MyString: string[100];
```

Hier wird die Variable **MyString** mit einer maximalen Länge von 100 Zeichen erstellt. Die folgenden Deklarationen sind mit der obigen Zeile identisch:

```

type CString = string[100];
var MyString: CString;

```

Bei Variablen, die auf diese Weise deklariert werden, wird dem Typ nur so viel Speicherplatz zugewiesen, wie für die angegebene Länge plus ein Byte erforderlich ist. Im obigen Beispiel belegt **MyString** 101 Byte. Für eine Variable des vordefinierten Typs **ShortString** wären dagegen 256 Byte erforderlich.

Bei einer Wertzuweisung an eine kurze String-Variable wird der String abgeschnitten, wenn die maximale Länge für den Typ überschritten wird.

Die Standardfunktionen **High** und **Low** bearbeiten Variablen und Typbezeichner für kurze Strings. **High** liefert die maximale Länge des kurzen String-Typs, während **Low** Null zurückgibt.

Lange String-Typen

Der Typ **AnsiString** (auch als langer String bezeichnet) stellt einen dynamisch zugewiesenen String dar, dessen maximale Länge nur durch den verfügbaren Speicherplatz begrenzt wird.

Eine **AnsiString**-Variable ist ein Zeiger, der vier Byte Speicherplatz belegt. Wenn die Variable leer ist (also einen String der Länge Null enthält), hat der Zeiger den Wert **nil**, und der String belegt keinen Speicherplatz. Ist die Variable nicht leer, zeigt sie auf einen dynamisch zugewiesenen Speicherblock, der einen String-Wert enthält. Die acht Byte vor der Stelle enthalten eine Längenangabe und einen Referenzzähler von je 32 Bit. Da dieser Speicherplatz auf dem Heap reserviert und vollkommen automatisch verwaltet wird, erfordert er keinerlei Benutzercode.

Da es sich bei **AnsiString**-Variablen um Zeiger handelt, können zwei oder mehrere dieser Variablen auf denselben Wert zeigen, ohne zusätzlichen Speicherplatz zu belegen. Der Compiler nützt dies zur Einsparung von Ressourcen. Auch Zuweisungen werden schneller ausgeführt. Sobald eine **AnsiString**-Variable freigegeben oder mit einem neuen Wert belegt wird, wird der

Referenzzähler des alten Strings (d.h. des vorhergehenden Wertes der Variablen) erniedrigt und der Referenzzähler des neuen Wertes (falls ein solcher zugewiesen wurde) erhöht. Erreicht der Referenzzähler eines Strings den Wert Null, wird der belegte Speicherplatz freigegeben. Dieser Vorgang wird als Referenzzählung bezeichnet. Wenn der Wert eines einzelnen Zeichens im String über einen Index geändert werden soll, wird eine Kopie des Strings angelegt. Dies ist aber nur möglich, wenn der betreffende Referenzzähler größer als 1 ist. Diesen Vorgang nennt man Copy-on-Write-Semantik.

WideString

Der Typ **WideString** repräsentiert einen dynamisch zugewiesenen String mit 16-Bit-Unicode-Zeichen. Er ähnelt in vielerlei Hinsicht dem Typ **AnsiString**. In Win32 ist der Typ **WideString** kompatibel mit dem COM-Typ BSTR.

Anmerkung: Unter Win32 unterliegen **WideString**-Werte nicht der Referenzzählung.

Win32 unterstützt Einzelbyte- und Multibyte-Zeichensätze sowie den Unicode-Zeichensatz. Bei einem Einzelbyte-Zeichensatz (SBCS = Single-Byte Character Set) repräsentiert jedes Byte eines Strings ein Zeichen.

In einem Multibyte-Zeichensatz (MBCS = Multi-Byte Character Set) werden einige Zeichen mit einem einzelnen Byte und andere mit mehreren Byte dargestellt. Das erste Byte eines Multibyte-Zeichens wird als führendes Byte bezeichnet. Im Allgemeinen stimmen die ersten 128 Zeichen eines Multibyte-Zeichensatzes mit den 7-Bit-ASCII-Zeichen überein. Jedes Byte, dessen Ordinalwert größer ist als 127, fungiert als führendes Byte eines Multibyte-Zeichens. Der Nullwert (#0) ist stets ein Einzelbyte-Zeichen. Multibyte-Zeichensätze, insbesondere Doppelbyte-Zeichensätze (DBCS = Double-Byte Character Set), werden in erster Linie für asiatische Sprachen verwendet.

Im Unicode-Zeichensatz wird jedes Zeichen mit zwei Byte dargestellt. Ein Unicode-String ist demnach eine Folge von Words und nicht von einzelnen Bytes. Unicode-Zeichen und -Strings werden auch als Wide-Zeichen bzw. WideStrings bezeichnet. Die ersten 256 Unicode-Zeichen stimmen mit dem ANSI-Zeichensatz überein. Windows unterstützt Unicode (UCS-2).

Über die Typen **Char**, **PChar**, **AnsiChar**, **PAnsiChar** und **AnsiString** unterstützt Delphi sowohl Einzelbyte- als auch Multibyte-Zeichen und -Strings. Eine zuverlässige Indizierung von Multibyte-Strings ist nicht möglich, da **S[i]** das *i*-te Byte (und nicht notwendigerweise das *i*-te Zeichen) in **S** darstellt. Jedoch gibt es für alle Standardfunktionen zur Stringverarbeitung multibytefähige Entsprechungen, die auch die Besonderheiten länderspezifischer Zeichensätze berücksichtigen. Die Namen von Multibyte-Funktionen beginnen normalerweise mit dem Wort **Ansi**. Beispielsweise trägt die Multibyte-Version von **StrPos** den Namen **AnsiStrPos**. Die Unterstützung von Multibyte-Zeichen ist betriebssystemabhängig und basiert auf dem verwendeten Gebietsschema.

Delphi unterstützt Unicode-Zeichen und -Strings über die Typen **WideChar**, **PWideChar** und **WideString**.

Nullterminierte Strings

In vielen Programmiersprachen, z. B. in C und C++, fehlt ein spezieller String-Datentyp. Diese Sprachen und die mit ihnen programmierten Umgebungen verwenden so genannte nullterminierte Strings. Ein nullterminierter String ist ein Zeichen-Array, dessen Index bei 0 beginnt und das mit einer NULL (#0) endet. Da das Array keine Längenangabe hat, wird das Ende des Strings durch das erste NULL-Zeichen markiert. Die Verarbeitung nullterminierter Strings erfolgt in Delphi mithilfe von Routinen, die sich in der Unit **SysUtils** befinden (siehe Standardroutinen und E/A (siehe Seite 989)). Dies ist beispielsweise erforderlich, wenn Sie Daten gemeinsam mit Systemen benutzen, die nullterminierte Strings verwenden.

Hier einige Beispiele für die Deklaration von Typen, die nullterminierte Strings speichern können:

```
type
  TIdentifier = array[0..15] of Char;
  TFileName = array[0..259] of Char;
  TMemoText = array[0..1023] of WideChar;
```

Wenn die erweiterte Syntax aktiviert ist (**{\$X+}**), können Sie einem statisch zugewiesenen Zeichen-Array, dessen Index bei 0 beginnt, eine String-Konstante zuweisen. Mit einem dynamischen Array ist dies nicht möglich. Wenn Sie eine Array-Konstante mit einem String initialisieren, der kürzer als die deklarierte Länge des Arrays ist, werden die verbleibenden Zeichen auf #0 gesetzt.

Zeiger, Arrays und String-Konstanten

Bei der Bearbeitung von nullterminierten Strings werden oft Zeiger benötigt. (Siehe Zeiger und Zeigertypen (siehe Seite 911).) String-Konstanten sind zuweisungskompatibel zu den Typen **PChar** und **PWideChar**, die Zeiger auf nullterminierte Arrays mit **Char**- und **WideChar**-Werten darstellen. Zum Beispiel:

```
var P: PChar;
  ...
P := 'Hello world!'
```

P zeigt auf einen Speicherbereich, der eine nullterminierte Kopie von "Hello world!" enthält. Dieses Verfahren ist zum Folgenden äquivalent:

```
const TempString: array[0..12] of Char = 'Hello world!';
var P: PChar;
  ...
P := @TempString[0];
```

Sie können String-Konstanten auch an Funktionen übergeben, die Wert- oder **const**-Parameter des Typs **PChar** oder **PWideChar** akzeptieren, z. B. `StrUpper('Hello world!')`. Der Compiler erzeugt (wie bei Zuweisungen an **PChar**) eine nullterminierte Kopie des Strings und übergibt der Funktion einen Zeiger auf diese Kopie. Außerdem können Sie **PChar**- und **PWideChar**-Konstanten einzeln oder in strukturierten Typen mit String-Literalen initialisieren. Beispiele:

```
const
  Message: PChar = 'Programm beendet';
  Prompt: PChar = 'Werte eingeben: ';
  Digits: array[0..9] of PChar = ('Null', 'Eins', 'Zwei', 'Drei', 'Vier', 'Fünf', 'Sechs',
  'Sieben', 'Acht', 'Neun');
```

Zeichen-Arrays, deren Index bei 0 beginnt, sind mit **PChar** und **PWideChar** kompatibel. Wenn Sie anstelle eines Zeigerwertes ein Zeichen-Array verwenden, wandelt der Compiler das Array in eine Zeiger-Konstante um, deren Wert der Adresse des ersten Elements im Array entspricht. Zum Beispiel:

```
var
  MyArray: array[0..32] of Char;
  MyPointer: PChar;
begin
  MyArray := 'Hello';
  MyPointer := MyArray;
  SomeProcedure(MyArray);
  SomeProcedure(MyPointer);
end;
```

Dieser Programmcode ruft `SomeProcedure` zweimal mit demselben Wert auf.

Ein Zeichenzeiger kann wie ein Array indiziert werden. Im obigen Beispiel gibt `MyPointer[0]` den Wert H zurück. Der Index legt einen Offset fest, der dem Zeiger vor der Dereferenzierung hinzugefügt wird. Bei **PWideChar**-Variablen wird der Index automatisch mit zwei multipliziert. Wenn es sich bei P um einen Zeichenzeiger handelt, ist `P[0]` identisch mit `P^` und bezeichnet das erste Zeichen im Array, `P[1]` das zweite Zeichen usw. `P[-1]` bezeichnet das "Zeichen", das unmittelbar links neben `P[0]` steht. Der Compiler führt für diese Indizes aber keine Bereichsprüfung durch.

Das folgende Beispiel zeigt anhand der Funktion `StrUpper`, wie unter Verwendung der Zeigerindizierung ein nullterminierter String durchlaufen wird:

```
function StrUpper(Dest, Source: PChar; MaxLen: Integer): PChar;
var
  I: Integer;
begin
  I := 0;
  while (I < MaxLen) and (Source[I] <> #0) do
  begin
    Dest[I] := UpCase(Source[I]);
    Inc(I);
  end;
end;
```

```

end;
Dest[I] := #0;
Result := Dest;
end;

```

Kombinieren von Delphi-Strings und nullterminierten Strings

Lange Strings (**AnsiString**-Werte) und nullterminierte Strings (**PChar**-Werte) lassen sich in Ausdrücken und Zuweisungen kombinieren. Außerdem können **PChar**-Werte an Funktionen und Prozeduren übergeben werden, die **AnsiString**-Parameter akzeptieren. Die Zuweisung `S := P`, in der `S` eine String-Variable und `P` ein **PChar**-Ausdruck ist, kopiert einen nullterminierten String in einen langen String.

Wenn in einer binären Operation der eine Operand ein langer String und der andere ein **PChar**-Ausdruck ist, wird der **PChar**-Operand in einen langen String umgewandelt.

Sie können einen **PChar**-Wert als langen String verwenden. Dies ist hilfreich, wenn eine String-Operation für zwei **PChar**-Werte durchgeführt werden soll. Zum Beispiel:

```
S := string(P1) + string(P2);
```

Es ist auch möglich, einen langen String in einen nullterminierten String umzuwandeln. Für Zeiger-Operatoren gelten die folgenden Regeln:

- Wenn `S` ein **AnsiString**-Ausdruck ist, wird `S` mit `PChar(S)` in einen nullterminierten String umgewandelt. Das Ergebnis ist ein Zeiger auf das erste Zeichen in `S`. Wenn es sich beispielsweise bei `Str1` und `Str2` um lange Strings handelt, können Sie die Win32 API-Funktion `MessageBox` folgendermaßen aufrufen: `MessageBox(0, PChar(Str1), PChar(Str2), MB_OK);`
- Außerdem haben Sie die Möglichkeit, mit `Pointer(S)` einen langen String in einen untypisierten Zeiger umzuwandeln. Wenn `S` allerdings leer ist, ergibt die Umwandlung **nil**.
- `PChar(S)` gibt stets einen Zeiger auf einen Speicherblock zurück. Wenn `S` leer ist, wird ein Zeiger auf `#0` zurückgegeben.
- Wenn Sie eine **AnsiString**-Variable in einen Zeiger konvertieren, bleibt dieser gültig, bis die Variable den Gültigkeitsbereich verlässt oder ihr ein neuer Wert zugewiesen wird. Wenn Sie einen beliebigen anderen **AnsiString**-Ausdruck in einen Zeiger umwandeln, ist der Zeiger nur innerhalb der Anweisung gültig, in der die Typumwandlung durchgeführt wird.
- Nach der Konvertierung eines **AnsiString**-Ausdrucks in einen Zeiger sollten Sie den Zeiger als schreibgeschützt ansehen. Der String kann über den Zeiger nur dann gefahrlos geändert werden, wenn folgende Bedingungen erfüllt sind:
 - Der Ausdruck, der umgewandelt werden soll, ist eine **AnsiString**-Variable.
 - Der String ist nicht leer.
 - Der String ist eindeutig, d. h. der Referenzzähler hat den Wert 1. Um sicherzustellen, dass der String eindeutig ist, rufen Sie eine der Prozeduren `SetLength`, `SetString` oder `UniqueString` auf.
 - Der String wurde seit der letzten Typumwandlung nicht geändert.
 - Die zu ändernden Zeichen befinden sich alle innerhalb des Strings. Sie dürfen für den Zeiger auf keinen Fall einen Index verwenden, der außerhalb des Bereichs liegt.

Diese Regeln gelten auch, wenn Sie **WideString**-Werte mit **PWideChar**-Werten kombinieren.

Siehe auch

Datentypen ([siehe Seite 887](#))

Einfache Typen ([siehe Seite 889](#))

Strukturierte Typen ([siehe Seite 902](#))

Zeiger und Zeigertypen ([siehe Seite 911](#))

Prozedurale Typen ([siehe Seite 914](#))

Variante Typen ([siehe Seite 917](#))

Kompatibilität und Identität von Typen (siehe Seite 921)

Typdeklaration (siehe Seite 923)

Variablen (siehe Seite 924)

Deklarierte Konstanten (siehe Seite 926)

4.1.1.4.4 Strukturierte Typen

Die Instanzen eines strukturierten Typs enthalten mehrere Werte. Zu den strukturierten Typen gehören Mengen-, Array-, Record- und Datei-, Klassen-, Klassenreferenz- und Interface-Typen. Mit Ausnahme von Mengen, die nur ordinale Werte enthalten, können strukturierte Typen auch andere strukturierte Typen beinhalten. Ein Typ kann beliebig viele strukturelle Ebenen umfassen.

Hinweis: Typisierte und nicht typisierte Dateitypen werden im .NET-Framework nicht unterstützt.

Per Voreinstellung sind die Werte in einem strukturierten Typ in einem Word- oder Double-Word-Raster ausgerichtet, um den Zugriff zu beschleunigen. Wenn Sie einen strukturierten Typ deklarieren, können Sie das reservierte Wort **packed** einfügen, um die Daten in komprimierter Form zu speichern: Ein Beispiel: `type TNumbers = packed array [1..100] of Real;`

Die Verwendung von **packed** verlangsamt den Zugriff auf die Daten. Im Falle eines Zeichen-Arrays beeinflusst **packed** auch die Kompatibilität der Typen. Weitere Informationen finden Sie unter Speicherverwaltung (siehe Seite 1017).

Dieses Thema enthält Informationen zu folgenden strukturierten Typen:

- Mengentypen
- Array-Typen (statisch und dynamisch)
- Record-Typen
- Dateitypen

Mengentypen

Eine Menge setzt sich aus mehreren Werten desselben ordinalen Typs zusammen. Die Werte haben keine feste Reihenfolge. Wenn ein Wert in einer Menge doppelt vorkommt, hat jedes Vorkommen dieselbe Bedeutung.

Der Bereich eines Mengentyps ist die Potenzmenge eines bestimmten Ordinaltyps, der als Basistyp bezeichnet wird. Die möglichen Werte eines Mengentyps sind Teilmengen des Basistyps, einschließlich der leeren Menge. Der Basistyp darf aus maximal 256 Werten bestehen. Die Ordinalpositionen der Werte müssen zwischen 0 und 255 liegen. Alle Konstruktionen der Form

`set of Basistyp`

bezeichnen einen Mengentyp. Dabei ist *Basistyp* ein entsprechender ordinaler Typ.

Aufgrund der Größenbeschränkung von Basistypen werden Mengentypen normalerweise mit Teilmengen definiert. So lässt sich mit den Deklarationen

```
type
  TSomeInts = 1..250;
  TIntSet = set of TSomeInts;
```

ein Mengentyp namens `TIntSet` erstellen, dessen Werte Integer-Zahlen im Bereich zwischen 1 und 250 sind. Sie können dazu auch folgende Anweisung verwenden:

```
type TIntSet = set of 1..250;
```

Mit dieser Deklaration können Sie beispielsweise folgende Mengen erstellen:

```
var Set1, Set2: TIntSet;
  ...
```

```
Set1 := [1, 3, 5, 7, 9];
Set2 := [2, 4, 6, 8, 10]
```

Die Konstruktion `set of ...` kann direkt in Variablendeklarationen verwendet werden:

```
var MySet: set of 'a'...'z';
...
MySet := ['a','b','c'];
```

Hier einige weitere Beispiele für Mengentypen:

```
set of Byte
set of (Club, Diamond, Heart, Spade)
set of Char;
```

Der Operator `in` überprüft, ob ein Element zu einer Menge gehört:

```
if 'a' in MySet then ... { Ausführung einer Aktion } ;
```

Jeder Mengentyp kann die leere Menge enthalten, die mit `[]` gekennzeichnet wird.

Arrays

Ein Array ist eine indizierte Menge von Elementen desselben Typs (des so genannten Basistyps). Da jedes Element einen eindeutigen Index hat, kann ein Array (im Gegensatz zu einer Menge) denselben Wert mehrmals und mit unterschiedlicher Bedeutung enthalten. Arrays können *statisch* oder *dynamisch* zugewiesen werden.

Statische Arrays

Statische Array-Typen werden mit der folgenden Konstruktion definiert:

```
array[Indextyp1, ..., Indextypn] of Basistyp;
```

Indextyp ist immer ein ordinaler Typ, dessen Bereich 2 GB nicht überschreitet. Da das Array über den *Indextyp* indiziert wird, ist die Anzahl der Elemente durch den angegebenen *Indextyp* beschränkt. In der Praxis sind die Indextypen normalerweise Integer-Teilbereiche.

Im einfachsten Fall eines eindimensionalen Arrays ist nur ein einziger Indextyp vorhanden. Beispiel:

```
var MyArray: array [1..100] of Char;
```

Hier wird eine Variable namens `MyArray` deklariert, die 100 Zeichenwerten enthält. Aufgrund dieser Deklaration bezeichnet `MyArray[3]` das dritte Zeichen in `MyArray`. Wenn Sie einen statischen Array erstellen und nicht allen Elementen Werte zuweisen, werden die unbenutzten Elemente dennoch vergeben. Diese Elemente enthalten zu Beginn zufällige Daten und sind mit nicht initialisierten Variablen vergleichbar.

Ein mehrdimensionales Array ist ein Array, der andere Arrays enthält. Zum Beispiel ist die Anweisung:

```
type TMatrix = array[1..10] of array[1..50] of Real;
```

äquivalent zu

```
type TMatrix = array[1..10, 1..50] of Real;
```

Unabhängig von der Art der Deklaration repräsentiert `TMatrix` immer ein Array mit 500 reellen Werten. Eine Variable namens `MyMatrix` vom Typ `TMatrix` kann auf zwei Arten indiziert werden: `MyMatrix[2,45]`; oder wie folgt: `MyMatrix[2][45]`. Genauso ist

```
packed array[Boolean, 1..10, TShoeSize] of Integer;
```

äquivalent zu

```
packed array[Boolean] of packed array[1..10] of packed array[TShoeSize] of Integer;
```

Die Standardfunktionen `Low` und `High` können auf Array-Typbezeichner und Variablen angewendet werden. Sie liefern die untere und obere Grenze des ersten Indextyps im Array. Die Standardfunktion `Length` gibt die Anzahl der Elemente in der ersten Dimension des Arrays zurück.

Ein eindimensionales, gepacktes statisches Array mit **Char**-Werten wird als gepackter String bezeichnet. Gepackte String-Typen sind mit String-Typen und anderen gepackten String-Typen kompatibel, die dieselbe Anzahl von Elementen haben. Weitere Informationen finden Sie unter Kompatibilität und Identität von Typen (siehe Seite 921).

Ein Array der Form `array[0..x] of Char` wird als nullbasiertes Zeichen-Array bezeichnet, da sein Index bei 0 beginnt. Nullbasierte Zeichen-Arrays werden zum Speichern von nullterminierten Strings verwendet und sind kompatibel mit **PChar**-Werten. Informationen hierzu finden Sie unter Nullterminierte Strings (siehe Seite 896).

Dynamische Arrays

Dynamische Arrays haben keine feste Größe oder Länge. Der Speicher für ein dynamisches Array wird reserviert, sobald Sie dem Array ein Wert zuweisen oder es an die Prozedur `SetLength` übergeben. Dynamische Array-Typen werden folgendermaßen deklariert:

```
array of Basistyp
```

Zum Beispiel deklariert die Anweisung:

```
var MyFlexibleArray: array of Real;
```

ein eindimensionales Array mit Elementen vom Typ `Real`. Diese Deklaration weist `MyFlexibleArray` keinen Speicherplatz zu. Um ein Array im Speicher anzulegen, rufen Sie `SetLength` auf. Nach der vorherigen Typdeklaration weist

```
SetLength(MyFlexibleArray, 20);
```

ein Array mit 20 reellen Zahlen und einem Index von 0 bis 19 zu. Dynamische Arrays haben immer einen Integer-Index, der bei 0 beginnt.

Dynamische Array-Variablen sind implizit Zeiger und werden mit derselben Referenzzählung verwaltet wie lange Strings. Um ein dynamisches Array freizugeben, weisen Sie einer Variablen, die das Array referenziert, den Wert **nil** zu, oder Sie übergeben die Variable an `Finalize`. Beide Methoden geben das Array unter der Voraussetzung frei, dass keine weiteren Referenzen darauf vorhanden sind. Dynamische Arrays werden immer freigegeben, sobald ihr Referenzzähler auf null fällt. Dynamische Arrays der Länge 0 haben immer den Wert **nil**. Verwenden Sie für dynamische Array-Variablen nicht den Dereferenzierungsoperator (^), und übergeben Sie sie auch nicht an die Prozeduren `New` oder `Dispose`.

Wenn `X` und `Y` Variablen desselben dynamischen Array-Typs sind, führt die Anweisung `X := Y` dazu, dass `X` auf dasselbe Array wie `Y` zeigt. (Es ist nicht erforderlich, vor dieser Operation Speicher für `X` zu reservieren.) Im Gegensatz zu Strings und statischen Arrays wird `copy-on-write` nicht für dynamische Arrays verwendet. Deshalb werden diese nicht automatisch kopiert, bevor einem ihrer Elemente ein Wert zugewiesen wird. Beispielsweise hat `A[0]` nach der Ausführung des folgenden Quelltextes den Wert 2:

```
var
  A, B: array of Integer;
begin
  SetLength(A, 1);
  A[0] := 1;
  B := A;
  B[0] := 2;
end;
```

Wenn `A` und `B` statische Arrays wären, hätte `A[0]` immer noch den Wert 1.

Durch Zuweisungen an ein dynamisches Array über den Index (wie beispielsweise `MyFlexibleArray[2] := 7`) wird für das Array kein neuer Speicherplatz reserviert. Der Compiler akzeptiert auch Indizes, die außerhalb des angegebenen Bereichs liegen.

Wenn Sie im Gegensatz dazu eine unabhängige Kopie eines dynamischen Arrays erstellen möchten, müssen Sie die globale Funktion `Copy` verwenden:

```
var
  A, B: array of Integer;
begin
```

```

SetLength(A, 1);
A[0] := 1;
B := Copy(A);
B[0] := 2; { B[0] <> A[0] }
end;

```

Bei einem Vergleich von dynamischen Array-Variablen werden nicht die Array-Werte, sondern die Referenzen verglichen. Deshalb liefert `A = B` nach Ausführung des Quelltextes

```

var
  A, B: array of Integer;
begin
  SetLength(A, 1);
  SetLength(B, 1);
  A[0] := 2;
  B[0] := 2;
end;

```

den Wert **False**, während `A[0] = B[0]` **True** zurückgibt.

Um ein dynamisches Array abzuschneiden, übergeben Sie es an `SetLength` oder `Copy` und weisen das Ergebnis wieder der Array-Variablen zu. Die Prozedur `SetLength` ist normalerweise schneller. Wenn `A` beispielsweise ein dynamisches Array ist, können Sie mit der Anweisung `A := SetLength(A, 0, 20)` die ersten 20 Elemente von `A` beibehalten und den Rest abschneiden.

Sobald einem dynamischen Array Speicherplatz zugewiesen wurde, kann es an die Standardfunktionen `Length`, `High` und `Low` übergeben werden. `Length` liefert die Anzahl der Elemente im Array, `High` den höchsten Index des Arrays (`Length - 1`) und `Low` den Wert 0. Bei einem Array mit der Länge Null liefert `High` das Ergebnis 1 (mit der unsinnigen Folge, dass `High` kleiner als `Low` ist).

Anmerkung: In einigen Funktions- und Prozedurdeklarationen werden Array-Parameter in der Form `array of Basistyp` ohne definierten Indextyp angegeben: `function CheckStrings(A: array of string): Boolean;`

In diesem Fall kann die Funktion auf alle Arrays des angegebenen Basistyps angewendet werden, unabhängig von der Größe der Arrays und der Art ihrer Indizierung. Es spielt auch keine Rolle, ob den Arrays der Speicherplatz statisch oder dynamisch zugewiesen wird.

Mehrdimensionale dynamische Arrays

Zur Deklaration von mehrdimensionalen dynamischen Arrays verwenden Sie aufeinander folgende `array of ...`-Konstruktionen. Zum Beispiel:

```

type TMessageGrid = array of array of string;
var Msgs: TMessageGrid;

```

Hier wird ein zweidimensionales String-Array deklariert. Um dieses Array zu instantiieren, rufen Sie `SetLength` mit zwei Integer-Argumenten auf. Wenn beispielsweise `I` und `J` Integer-Variablen sind, wird Speicherplatz für ein `I mal J` großes Array zugewiesen:

```
SetLength(Msgs, I, J);
```

`Msgs[0,0]` bezeichnet dann ein Element dieses Arrays.

Sie können auch mehrdimensionale dynamische Arrays anlegen, die nicht gleichförmig sind. Rufen Sie dazu als Erstes die Funktion `SetLength` auf, und übergeben Sie ihr Parameter für die ersten `n` Dimensionen des Arrays. Zum Beispiel:

```

var Ints: array of array of Integer;
SetLength(Ints, 10);

```

Mit dieser Anweisung weisen Sie dem Array `Ints` Speicherplatz für zehn Zeilen zu. Den Speicher für die Spalten können Sie später einzeln zuweisen (und dabei unterschiedliche Längen angeben):

```
SetLength(Ints[2], 5);
```

Die dritte Spalte von `Ints` kann damit fünf Integer-Werte aufnehmen, und Sie können ihr nun Werte zuweisen (auch wenn die anderen Spalten nicht zugewiesen sind), z. B. `Ints[2,4] := 6`.

Im folgenden Beispiel wird mithilfe von dynamischen Arrays (und der in der Unit `SysUtils` deklarierten Funktion `IntToStr`) eine ungleichförmige String-Matrix erstellt:

```
4
var
  A: array of array of string;
  I, J : Integer;
begin
  SetLength(A, 10);
  for I := Low(A) to High(A) do
    begin
      SetLength(A[I], I);
      for J := Low(A[I]) to High(A[I]) do
        A[I,J] := IntToStr(I) + ',' + IntToStr(J) + ' ';
    end;
end;
```

Array-Typen und Zuweisungen

Arrays sind nur dann zuweisungskompatibel, wenn sie denselben Typ haben. Da Delphi Namensäquivalente für Typen verwendet, wird folgender Quelltext nicht kompiliert:

```
var
  Int1: array[1..10] of Integer;
  Int2: array[1..10] of Integer;
  ...
  Int1 := Int2;
```

Damit die Zuweisung korrekt bearbeitet werden kann, deklarieren Sie die Variablen folgendermaßen:

```
var Int1, Int2: array[1..10] of Integer;
```

oder

```
type IntArray = array[1..10] of Integer;
var
```

```
  Int1: IntArray;
  Int2: IntArray;
```

Dynamisch zugewiesene, multidimensionale Arrays (.NET)

Auf der .NET-Plattform können multidimensionale Arrays mit der Standardfunktion `New` dynamisch zugewiesen werden. In der Syntax von `New` wird bei der Array-Deklaration die Anzahl der Dimensionen, nicht aber deren eigentliche Größe festgelegt. Sie übergeben dann den Elementtyp, die tatsächlichen Array-Dimensionen oder eine Liste zum Initialisieren des Array an die Funktion `New`. Die Syntax der Array-Deklaration lautet:

```
array[, ..., ] of Basistyp;
```

In der Syntax wird für die Anzahl der Dimensionen ein Komma als Platzhalter verwendet; die tatsächliche Größe wird erst zur Laufzeit durch den Aufruf der Funktion `New` festgelegt. Es gibt zwei Formen der Funktion `New`: Die erste übernimmt den Elementtyp und die Größe des Array, die zweite den Elementtyp und eine Liste zum Initialisieren des Array. Im folgenden Code werden beide Formen demonstriert:

```
var
  a: array [,] of integer; // dreidimensionales Array
  b : array [,] of integer; // zweidimensionales Array
  c : array [,] of TPoint; // zweidimensionales TPoint-Array

begin
```

```

a := New(array[3,5,7] of integer);           // New übernimmt den Elementtyp und die
Größe jeder Dimension.
b := New(array[,] of integer, ((1,2,3), (4,5,6))); // New übernimmt den Elementtyp und die
Initialisierungsliste.                      // New übernimmt die
TPoint-Initialisierungsliste.
c := New(array[,] of TPoint, (((X:1;Y:2), (X:3;Y:4)), ((X:5;Y:6), (X:7;Y:8)))); 
end.

```

Sie können das Array zuweisen, indem Sie Variablen- oder Konstantenausdrücke an die Funktion New übergeben:

```

var
  a: array[,] of integer;
  r,c: Integer;

begin
  r := 4;
  c := 17;

  a := New(array [r,c] of integer);

```

Sie können das Array auch mit der Prozedur SetLength zuweisen und den Array-Ausdruck sowie die Größe der Dimensionen übergeben:

```

var
  a: array[,] of integer;
  b : array[,,] of integer;

begin
  SetLength(a, 4,5);
  SetLength(b, 3,5,7);
end.

```

Mit der Funktion Copy kann eine Kopie des gesamten Array angelegt werden. Mit Copy kann kein teilweises Duplikat eines Array erstellt werden.

Ein dynamisch zugewiesenes, rechteckiges Array kann nicht an die Funktionen Low und High übergeben werden. Dies erzeugt einen Compilerfehler.

Records (traditionelle)

Ein **Record** (in einigen Programmiersprachen auch als Struktur bezeichnet) stellt eine heterogene Menge von Elementen dar. Die Elemente werden Felder genannt. In der Deklaration eines Record-Typs wird für jedes Feld ein Name und ein Typ festgelegt. Die Syntax für die Deklaration eines Record-Typs lautet

```

type RecordTypName = record
  Feldlistel: Typ1;
  ...
  Feldlisten: Typn;
end;

```

Recordtypname ist ein gültiger Bezeichner, *Typ* gibt einen Typ an, und *Feldliste* ist ein gültiger Bezeichner oder eine Liste von Bezeichnern, die durch Kommas voneinander getrennt sind. Der letzte Strichpunkt ist optional.

Die folgende Deklaration legt einen Record-Typ namens TDateRec an:

```

type
  TDateRec = record
    Year: Integer;
    Month: (Jan, Feb, Mar, Apr, May, Jun,
             Jul, Aug, Sep, Oct, Nov, Dec);
    Day: 1..31;
  end;

```

Jeder TDateRec-Record enthält drei Felder: einen Integer-Wert namens Year, einen Aufzählungswert namens Month und einen weiteren Integer-Wert zwischen 1 und 31 namens Day. Die Bezeichner Year, Month und Day sind Feldbezeichner für TDateRec und verhalten sich wie Variablen. Die Typdeklaration für TDateRec weist den Feldern Year, Month und Day aber

keinen Speicherplatz zu. Die Reservierung des Speichers erfolgt erst, wenn der Record instantiiert wird:

```
var Record1, Record2: TDateRec;
```

Diese Variablen Deklaration erzeugt zwei Instanzen von `TDateRec` namens `Record1` und `Record2`.

Sie können auf die Felder eines Records zugreifen, indem Sie die Feldbezeichner mit dem Record-Namen qualifizieren:

```
Record1.Year := 1904;
Record1.Month := Jun;
Record1.Day := 16;
```

Alternativ dazu ist auch die Verwendung einer **with**-Anweisung möglich:

```
with Record1 do
begin
  Year := 1904;
  Month := Jun;
  Day := 16;
end;
```

Nun können die Werte der Felder von `Record1` nach `Record2` kopiert werden:

```
Record2 := Record1;
```

Da der Gültigkeitsbereich eines Feldbezeichners auf den Record beschränkt ist, in dem er sich befindet, brauchen Sie nicht auf eventuelle Namenskonflikte zwischen Feldbezeichnern und anderen Variablen zu achten.

Statt der Definition von Record-Typen kann die Konstruktion `record ...` auch direkt in Variablen Deklarationen verwendet werden:

```
var S: record
  Name: string;
  Age: Integer;
end;
```

Eine solche Deklaration macht aber wenig Sinn, da der eigentliche Zweck eines Records darin besteht, die wiederholte Deklaration ähnlicher Variablen Gruppen zu vermeiden. Außerdem sind separat deklarierte Records auch dann nicht Zuweisungskompatibel, wenn ihre Strukturen identisch sind.

Variante Teile in Record-Typen

Ein Record-Typ kann einen varianten Teil enthalten, der einer **case**-Anweisung ähnelt. Dieser variante Teil muss in der Typ Deklaration nach den Feldern angegeben werden.

Mit der folgenden Syntax deklarieren Sie einen Record-Typ mit einem varianten Teil:

```
type RecordTypName = record
  Feldlistel: Typ1;
  ...
  Feldlisten: Typn;
  case Tag: Ordinaltyp of
    Konstantenlistel: (Variante1);
    ...
    Konstantenlisten: (Varianten);
  end;
```

Der erste Teil der Deklaration (bis zum reservierten Wort **case**) ist identisch mit der Deklaration eines Standard-Records. Der Rest der Deklaration (von **case** bis zum abschließenden optionalen Strichpunkt) stellt den varianten Teil mit folgenden Komponenten dar. Im varianten Teil

- Ist **Tag** optional und kann ein beliebiger, gültiger Bezeichner sein. Wenn Sie **Tag** weglassen, entfällt auch der Doppelpunkt (:) .
- **Ordinaltyp** bezeichnet einen ordinalen Typ.
- Jede **Konstantenliste** ist eine Konstante (oder eine Liste von Konstanten, die durch Kommas voneinander getrennt sind), die einen Wert des Typs **Ordinaltyp** bezeichnet. In allen **Konstantenlisten** darf jeder Wert nur einmal vorkommen.

- *Variante* ist eine Liste mit Deklarationen, die durch Strichpunkte voneinander getrennt sind. Die Liste ähnelt in etwa den *Feldliste*: Typ-Konstruktionen im Hauptteil des Record-Typs. Eine Variante hat demnach folgende Form:

```
Feldlistel: Typ1;
...
Feldlisten: Typn;
```

Dabei ist *Feldliste* ein gültiger Bezeichner oder eine Liste von Bezeichnern, die durch Kommas voneinander getrennt sind. *Typ* ist ein Typ. Der letzte Strichpunkt ist optional. Bei *Typ* darf es sich nicht um einen langen String, ein dynamisches Array, eine Variante (Typ *Variant*) oder ein Interface handeln. Strukturierte Typen, die lange Strings, dynamische Arrays, Varianten oder Interfaces enthalten, sind ebenfalls nicht zulässig. Zeiger auf diese Typen dürfen jedoch verwendet werden.

Die Syntax für Records mit varianten Teilen ist kompliziert, ihre Semantik ist jedoch einfach. Der variante Teil eines Records enthält mehrere Varianten, die sich denselben Speicherplatz teilen. Auf die Felder einer Variante kann jederzeit ein Lese- oder Schreibzugriff ausgeführt werden. Wenn Sie allerdings zunächst in ein Feld einer Variante schreiben und anschließend in ein Feld einer anderen Variante, kann das zum Überschreiben der eigenen Daten führen. Falls vorhanden, agiert *Tag* als gesondertes Feld (des Typs *Ordinal*) im nichtvarianten Teil des Records.

Variante Teile erfüllen zwei Funktionen, die sich am besten anhand eines Beispiels verdeutlichen lassen. Angenommen, Sie möchten einen Record-Typ erstellen, der Felder für unterschiedliche Daten enthält. Sie wissen, dass Sie nie alle Felder einer einzelnen Record-Instanz benötigen werden. Beispiel:

```
type
  TEmployee = record
    FirstName, LastName: string[40];
    BirthDate: TDate;
    case Salaried: Boolean of
      True: (AnnualSalary: Currency);
      False: (HourlyWage: Currency);
  end;
```

Diesem Beispiel liegt die Überlegung zugrunde, dass ein Angestellter entweder ein jährliches Festgehalt (*AnnualSalary*) oder einen Stundenlohn (*HourlyWage*) erhält, und dass für einen Angestellten immer nur eine der Zahlungsarten in Frage kommt. Wenn Sie eine Instanz von *TEmployee* anlegen, braucht also nicht für beide Felder Speicherplatz reserviert zu werden. In diesem Beispiel unterscheiden sich die Varianten nur durch die Feldnamen. Die Felder könnten aber auch unterschiedliche Typen haben. Hier einige komplexere Beispiele:

```
type
  TPerson = record
    FirstName, LastName: string[40];
    BirthDate: TDate;
    case Citizen: Boolean of
      True: (Birthplace: string[40]);
      False: (Country: string[20];
               EntryPort: string[20];
               EntryDate, ExitDate: TDate);
  end;

type
  TShapeList = (Rectangle, Triangle, Circle, Ellipse, Other);
  TFigure = record
    case TShapeList of
      Rectangle: (Height, Width: Real);
      Triangle: (Side1, Side2, Angle: Real);
      Circle: (Radius: Real);
      Ellipse, Other: ();
  end;
```

Der Compiler weist jeder Record-Instanz so viel Speicherplatz zu, wie die Felder in der größten Variante benötigen. Die optionalen Komponenten *Tag* und *Konstantenliste* (wie *Rectangle*, *Triangle* usw. im letzten Beispiel) spielen für die Art und Weise, wie der Compiler die Felder verwaltet, keine Rolle. Sie erleichtern lediglich die Arbeit des Programmierers.

Wie bereits erwähnt, erfüllen variante Teile noch eine zweite Aufgabe. Sie können dieselben Daten so behandeln, als würden sie

zu unterschiedlichen Typen gehören. Dies gilt auch in den Fällen, in denen der Compiler eine Typumwandlung nicht zulässt. Wenn beispielsweise das erste Feld einer Variante einen 64-Bit-**Real**-Typ und das erste Feld einer anderen Variante einen 32-Bit-**Integer**-Wert enthält, können Sie dem **Real**-Feld einen Wert zuweisen und anschließend die ersten 32 Bits als **Integer**-Wert verwenden (indem Sie sie beispielsweise an eine Funktion übergeben, die einen Integer-Parameter erwartet).

Records (erweiterte)

Zusätzlich zu den traditionellen Record-Typen lässt die Delphi-Sprache komplexere und "klassenähnliche" Record-Typen zu. Zu den Feldern können Records Eigenschaften und Methoden (einschließlich Konstruktoren), Klasseneigenschaften, Klassenmethoden, Klassenfelder und verschachtelte Typen haben. Weitere Informationen hierzu finden Sie in der Dokumentation zu Klassen und Objekten. Im Folgenden finden Sie eine Beispiel-Record-Typdefinition mit einigen "klassenähnlichen" Merkmalen.

```
4
type
  TMyRecord = record
    type
      TInnerColorType = Integer;
    var
      Red: Integer;
    class var
      Blue: Integer;
    procedure printRed();
    constructor Create(val: Integer);
    property RedProperty: TInnerColorType read Red write Red;
    class property BlueProp: TInnerColorType read Blue write Blue;
  end;

  constructor TMyRecord.Create(val: Integer);
begin
  Red := val;
end;

procedure TMyRecord.printRed;
begin
  writeln('Red: ', Red);
end;
```

Obwohl Records nun einige der Merkmale von Klassen besitzen, gibt es wichtige Unterschiede zwischen Klassen und Records.

- Records unterstützen keine Vererbung.
- Records können variante Teile enthalten; Klassen nicht.
- Records sind Wertetypen, daher werden Sie bei der Zuweisung kopiert, per Wert übergeben und dem Stack zugewiesen, wenn sie nicht als global definiert sind oder explizit mit den Funktionen **New** und **Dispose** zugewiesen werden. Klassen sind Referenztypen, daher werden Sie bei der Zuweisung nicht kopiert, per Referenz übergeben und dem Heap zugewiesen.
- Records ermöglichen das Überladen von Operatoren auf Win32- und .NET-Plattformen; Klassen ermöglichen das Überladen von Operatoren nur bei .NET.
- Records werden automatisch mit einem Standardkonstruktur ohne Argumente erzeugt, Klassen dagegen müssen explizit erzeugt werden. Weil Records einen argumentlosen Standardkonstruktur haben, muss jeder benutzerdefinierte Record-Konstruktur ein oder mehr Parameter haben.
- Record-Typen können keine Destruktoren haben.
- Virtuelle Methoden (die mit den Schlüsselwörtern **virtual**, **dynamic** und **message** angegeben werden) dürfen in Record-Typen nicht verwendet werden.
- Im Gegensatz zu Klassen können Record-Typen auf der Win32-Plattform keine Interfaces implementieren; auf der .NET-Plattform können Records jedoch Interfaces implementieren.

Dateitypen (Win32)

Dateitypen, die auf der Win32-Plattform zur Verfügung stehen, sind Folgen von Elementen desselben Typs. Für Standard-E/A-Routinen wird der vordefinierte Typ **TextFile** oder **Text** verwendet. Dieser Typ repräsentiert eine Datei, die in

Zeilen angeordnete Zeichen enthält. Ausführliche Informationen über die Datei-E/A finden Sie unter Standardroutinen und E/A (siehe Seite 989).

Die Deklaration eines Dateityps erfordert folgende Syntax:

```
type Dateitypname = file of Typ
```

Dateitypname ist ein gültiger Bezeichner. *Typ* ist ein Typ fester Länge. Zeiger sind weder als implizite noch als explizite Typen erlaubt. Dynamische Arrays, lange Strings, Klassen, Objekte, Zeiger, Varianten, andere Dateien oder strukturierte Typen, die einen dieser Typen beinhalten, können deshalb nicht in Dateien enthalten sein.

Zum Beispiel:

```
type
  PhoneEntry = record
    FirstName, LastName: string[20];
    PhoneNumber: string[15];
    Listed: Boolean;
  end;
  PhoneList = file of PhoneEntry;
```

Hier wird ein Dateityp für die Aufzeichnung von Namen und Telefonnummern deklariert.

Die Konstruktion *file of* ... kann direkt in einer Variablen Deklaration verwendet werden. Zum Beispiel:

```
var List1: file of PhoneEntry;
```

Das Wort **file** selbst gibt eine untypisierte Datei an:

```
var DataFile: file;
```

Weitere Informationen hierzu finden Sie unter Untypisierte Dateien (siehe Seite 989).

Dateitypen sind in Arrays und Records nicht erlaubt.

Siehe auch

Datentypen (siehe Seite 887)

Einfache Typen (siehe Seite 889)

String-Typen (siehe Seite 896)

Zeiger und Zeigertypen (siehe Seite 911)

Prozedurale Typen (siehe Seite 914)

Variante Typen (siehe Seite 917)

Kompatibilität und Identität von Typen (siehe Seite 921)

Typdeklaration (siehe Seite 923)

Variablen (siehe Seite 924)

Deklarierte Konstanten (siehe Seite 926)

Klassen und Objekte (siehe Seite 948)

4.1.1.4.5 Zeiger und Zeigertypen

Ein Zeiger ist eine Variable, die eine Speicheradresse angibt. Wenn ein Zeiger die Adresse einer anderen Variablen enthält, zeigt er auf die Position dieser Variablen im Speicher oder auf die Daten, die an dieser Position gespeichert sind. Bei einem Array oder einem anderen strukturierten Typ enthält der Zeiger die Adresse des ersten Elements der Struktur. Ist diese Adresse bereits vergeben, enthält der Zeiger die Adresse des ersten Elements.

Zeiger sind typisiert, d. h. sie geben die Art der Daten an, auf die sie zeigen. Der Allzwecktyp **Pointer** repräsentiert einen Zeiger auf beliebige Daten, während spezialisierte Zeigertypen nur auf bestimmte Datentypen zeigen. Zeiger belegen immer vier Byte Speicherplatz.

Dieses Thema enthält Informationen zu folgenden Bereichen:

- Allgemeiner Überblick über Zeigertypen
- Deklarieren und Verwenden der von Delphi unterstützten Zeigertypen

Zeiger im Überblick

Das folgende Beispiel zeigt, wie Zeiger funktionieren:

```
4
1 var
2     X, Y: Integer; // X und Y sind Integer-Variablen
3     P: ^Integer    // P zeigt auf einen Integer
4 begin
5     X := 17;       // Wertzuweisung an X
6     P := @X;       // Zuweisung der Adresse von X an P
7     Y := P^;       // Dereferenzierung von P; Zuweisung des Ergebnisses an Y
8 end;
```

In Zeile 2 werden **X** und **Y** als Variablen des Typs **Integer** deklariert. Zeile 3 deklariert **P** als Zeiger auf einen **Integer**-Wert. **P** kann also auf die Position von **X** oder **Y** zeigen. In Zeile 5 wird **X** ein Wert zugewiesen. Zeile 6 weist **P** die Adresse von **X** (angegeben durch **@X**) zu. Schließlich wird in Zeile 7 der Wert an der Adresse ermittelt, auf die **P** zeigt (angegeben durch **P^**), und **Y** zugewiesen. Nach der Ausführung dieses Programms haben **X** und **Y** denselben Wert (17).

Der Operator **@** wird hier verwendet, um die Adresse einer Variablen zu ermitteln. Sie können diesen Operator aber auch für Funktionen und Prozeduren einsetzen. Weitere Informationen hierzu finden Sie unter Der Operator **@** (siehe Seite 877) und Prozedurale Typen in Anweisungen und Ausdrücken (siehe Seite 914).

Wie das obige Beispiel zeigt, erfüllt das Symbol **^** zwei Funktionen. Es kann vor einem Typbezeichner stehen. Beispiel:

^Typname

In diesem Fall bezeichnet das Symbol einen Typ, der Zeiger auf Variablen des Typs **Typname** darstellt. Das Symbol kann aber auch auf eine Zeigervariable folgen:

Zeiger^

In diesem Fall dereferenziert das Symbol den Zeiger, liefert also den Wert an der Speicheradresse, die der Zeiger angibt.

Das obige Beispiel sieht auf den ersten Blick wie eine etwas umständliche Möglichkeit aus, den Wert einer Variablen in eine andere zu kopieren. Dies ließe sich viel einfacher durch eine entsprechende Zuweisung erreichen. Die Verwendung von Zeigern ist aber aus mehreren Gründen sinnvoll. Sie sind für das Verständnis der Sprache Delphi wichtig, da sie in einem Programm oft hinter den Kulissen agieren und nicht explizit auftreten. Zeiger werden von allen Datentypen verwendet, die große, dynamisch zugewiesene Speicherblöcke benötigen. Beispielsweise sind lange String-Variablen ebenso wie Klasseninstanzvariablen implizite Zeiger. In vielen komplexen Programmierkonstrukten ist die Verwendung von Zeigern unverzichtbar.

In vielen Situationen sind Zeiger die einzige Möglichkeit, die strikte Typisierung der Daten durch Delphi zu umgehen. Sie können z. B. die in einer Variablen gespeicherten Daten ohne Berücksichtigung ihres Typs verarbeiten, indem Sie sie über den Allzweckzeiger **Pointer** referenzieren, diesen in den gewünschten Typ umwandeln und ihn anschließend wieder dereferenzieren. Hier ein Beispiel, in dem die in einer reellen Variable gespeicherten Daten an eine Integer-Variable zugewiesen werden:

```
type
  PInteger = ^Integer;
var
  R : Single;
  I: Integer;
  P: Pointer;
  PI: PInteger;
```

```

begin
  ...
  P := @R;
  PI := PInteger(P);
  I := PI^;
end;

```

Reelle und Integer-Werte werden natürlich in unterschiedlichen Formaten gespeichert. Durch die Zuweisung werden lediglich die binären Rohdaten von `R` nach `I` kopiert. Eine Konvertierung findet dabei nicht statt.

Neben der Zuweisung des Ergebnisses einer `@`-Operation können aber auch diverse Standardroutinen eingesetzt werden, um einem Zeiger einen Wert zuzuweisen. Mit den Prozeduren `New` und `GetMem` lässt sich z. B. einem vorhandenen Zeiger eine Speicheradresse zuweisen. Die Funktionen `Addr` und `Ptr` geben einen Zeiger auf eine bestimmte Adresse oder Variable zurück.

Dereferenzierte Zeiger können qualifiziert werden und als Qualifizierer agieren. Ein Beispiel hierfür ist der Ausdruck `P1^.Data^`.

Das reservierte Wort **nil** ist eine spezielle Konstante, die jedem Zeiger zugewiesen werden kann. Ein solcher Zeiger verweist dann auf kein bestimmtes Ziel.

Zeigertypen

Mit der folgenden Syntax kann ein Zeiger auf jeden beliebigen Typ deklariert werden:

```
type Zeigertypname =^Typ
```

Bei der Definition eines Record-Typs oder eines anderen Datentyps könnte es hilfreich sein, auch einen Zeiger auf diesen Typ zu deklarieren. Dies erleichtert die Verarbeitung von Instanzen des Typs, da das Kopieren größerer Speicherblöcke entfällt.

Anmerkung: Sie können einen Zeigertyp definieren, bevor Sie den Typ deklarieren, auf den er zeigt.

Es gibt Standardzeigertypen für unterschiedliche Verwendungszwecke. Mit dem Allzwecktyp **Pointer** kann jeder Datentyp referenziert werden. Eine Dereferenzierung von Variablen des Typs **Pointer** ist nicht möglich. Den Versuch, einer **Pointer**-Variablen das Symbol `^` nachzustellen, weist der Compiler zurück. Wenn Sie auf Daten zugreifen wollen, auf die eine **Pointer**-Variable zeigt, müssen Sie diese Variable in einen anderen Zeigertyp umwandeln, bevor Sie sie dereferenzieren.

Zeiger auf Zeichen

Die beiden fundamentalen Zeigertypen **PAnsiChar** und **PWideChar** stellen Zeiger auf **AnsiChar**- bzw. **WideChar**-Werte dar. Der generische Typ **PChar** repräsentiert einen Zeiger auf einen **Char**-Wert (in der aktuellen Implementierung auf einen **AnsiChar**-Wert). Diese Zeigertypen auf Zeichen werden zur Verarbeitung von nullterminierten Strings eingesetzt. Informationen hierzu finden Sie unter Nullterminierte Strings (siehe Seite 896).

Typgeprüfte Zeiger

Die Compiler-Direktive `$_T` steuert die Typen der Zeigerwerte, die durch den Operator `@` generiert werden. Diese Direktive hat das folgende Format:

```
{$_T+} oder {$_T-}
```

Im Status `{$_T-}` ist der Ergebnistyp des Operators `@` immer ein untypisierter Zeiger, der mit allen anderen Zeigertypen kompatibel ist. Wird `@` auf eine Variablenreferenz im Status `{$_T+}` zugewiesen, lautet der Typ des Ergebnisses `^T`, wobei `T` nur mit den Zeigern auf den Typ der Variable kompatibel ist.

Weitere Standardzeigertypen

Viele der häufig verwendeten Standardzeigertypen sind in den Units `System` und `SysUtils` deklariert.

Eine Auswahl der in System und SysUtils deklarierten Zeigertypen

Zeigertyp	Zeigt auf Variablen des Typs
PAnsiString, PString	AnsiString
PByteArray	TByteArray (deklariert in <code>SysUtils</code>). Wird für die Typumwandlung dynamisch verwalteter Speicherblöcke verwendet, um Array-Zugriffe zu ermöglichen.
PCurrency, PDouble, PExtended, PSingle	Currency, Double, Extended, Single
PInteger	Integer
POleVariant	OleVariant
PShortString	ShortString . Ist bei der Portierung von veraltetem Code hilfreich, in dem noch der Typ PString verwendet wird.
PTTextBuf	TTextBuf (deklariert in <code>SysUtils</code>). TTextBuf ist der interne Puffer für den Datei-Record TTextRec .
PVarRec	TVarRec (deklariert in <code>System</code>)
PVariant	Variant
PWideString	WideString
PWordArray	TWordArray (deklariert in <code>SysUtils</code>). Wird für die Typumwandlung dynamisch verwalteter Speicherblöcke verwendet, wenn diese als Array mit 2-Byte-Werten bearbeitet werden sollen.

Siehe auch

Datentypen ([siehe Seite 887](#))

Einfache Typen ([siehe Seite 889](#))

String-Typen ([siehe Seite 896](#))

Strukturierte Typen ([siehe Seite 902](#))

Prozedurale Typen ([siehe Seite 914](#))

Variante Typen ([siehe Seite 917](#))

Kompatibilität und Identität von Typen ([siehe Seite 921](#))

Typdeklaration ([siehe Seite 923](#))

Variablen ([siehe Seite 924](#))

Deklarierte Konstanten ([siehe Seite 926](#))

4.1.1.4.6 Prozedurale Typen

Prozeduren und Funktionen können als Wert betrachtet und einer Variablen zugewiesen werden, so dass sie sich als Parameter übergeben lassen. Dieses Verfahren wird durch so genannte prozedurale Typen ermöglicht.

Allgemeines zu prozeduralen Typen

Prozeduren und Funktionen können als Wert betrachtet und einer Variablen zugewiesen werden, so dass sie sich als Parameter übergeben lassen. Dieses Verfahren wird durch so genannte prozedurale Typen ermöglicht. Nehmen wir an, Sie definieren eine Funktion mit dem Namen `Calc`, die zwei Integer-Parameter übernimmt und einen Integer-Wert zurückgibt:

```
function Calc(X,Y: Integer): Integer;
```

Sie können die `Calc`-Funktion nun der Variablen `F` zuweisen:

```
var F: function(X,Y: Integer): Integer;
```

```
F := Calc;
```

Wenn Sie im Kopf einer Prozedur oder Funktion den Bezeichner nach dem Wort **procedure** bzw. **function** entfernen, erhalten Sie den Namen eines prozeduralen Typs. Dieser Typname kann direkt in einer Variablen Deklaration (siehe oben) verwendet oder zur Deklaration neuer Typen benutzt werden:

```
type
  TIntegerFunction = function: Integer;
  TProcedure = procedure;
  TStrProc = procedure(const S: string);
  TMathFunc = function(X: Double): Double;
var
  F: TIntegerFunction;      { F ist eine parameterlose Funktion, die einen Integer zurückgibt }
  Proc: TProcedure;        { Proc ist eine parameterlose Prozedur }
  SP: TStrProc;            { SP ist eine Prozedur, die einen String-Parameter übernimmt }
  M: TMathFunc;             { M ist eine Funktion, die einen Double-Parameter (real) übernimmt
und einen Double-Wert zurückgibt }

  procedure FuncProc(P: TIntegerFunction); { FuncProc ist eine Prozedur, die als einzigen
Parameter eine parameterlose Integer-Funktion übernimmt }
```

In Win32 sind alle oben aufgeführten Variablen so genannte Prozedurenzeiger, also Zeiger auf die Adresse einer Prozedur oder Funktion. Auf der .NET-Plattform werden prozedurale Typen als Delegates implementiert. Um eine Methode eines Instanzobjekts zur referenzieren (siehe Klassen und Objekte (siehe Seite 948)), muss dem Namen des prozeduralen Typs die Klausel `of object` hinzugefügt werden. Beispiel:

```
type
  TMethod      = procedure of object;
  TNotifyEvent = procedure(Sender: TObject) of object;
```

Diese Typen sind Methodenzeiger. Ein Methodenzeiger wird in Form zweier Zeiger codiert, von denen der erste die Adresse der Methode speichert, während der zweite eine Referenz auf das Objekt enthält, zu dem die Methode gehört. Nach den Deklarationen

```
type
  TNotifyEvent = procedure(Sender: TObject) of object;
  TMainForm = class(TForm)
    procedure ButtonClick(Sender: TObject);
  ...
end;
var
  MainForm: TMainForm;
  OnClick: TNotifyEvent
```

ist die folgende Zuweisung korrekt:

```
OnClick := MainForm.ButtonClick;
```

Zwei prozedurale Typen sind kompatibel, wenn sie folgende Bedingungen erfüllen:

- Sie verwenden dieselbe Aufrufkonvention.
- Sie haben denselben (oder keinen) Rückgabetyp.
- Sie verwenden die gleiche Anzahl identischen Typs an den entsprechenden Positionen (die Parameternamen sind nicht von Bedeutung).

Unter Win32 sind prozedurale Zeigertypen immer inkompatibel mit Methodenzeigertypen; dies gilt nicht für die .NET-Plattform. Der Wert `nil` kann jedem prozeduralen Typ zugewiesen werden.

Verschachtelte Prozeduren und Funktionen (Routinen, die in anderen Routinen deklariert sind), können nicht als prozedurale Werte verwendet werden. Dasselbe gilt für vordefinierte Prozeduren und Funktionen (Standardroutinen). Wenn Sie eine Standardroutine wie `Length` als prozeduralen Wert verwenden wollen, müssen Sie die Routine gewissermaßen "verpacken":

```
function FLength(S: string): Integer;
begin
  Result := Length(S);
```

```
end;
```

Prozedurale Typen in Anweisungen und Ausdrücken

Wenn links in einer Zuweisung eine prozedurale Variable steht, erwartet der Compiler auf der rechten Seite einen prozeduralen Wert. Durch die Zuweisung wird aus der Variablen auf der linken Seite ein Zeiger auf die Funktion oder Prozedur auf der rechten Seite. In einem anderen Kontext führt die Verwendung einer prozeduralen Variable zu einem Aufruf der referenzierten Prozedur oder Funktion. Prozedurale Variablen können auch als Parameter übergeben werden:

```
var
  F: function(X: Integer): Integer;
  I: Integer;
  function SomeFunction(X: Integer): Integer;
  ...
  F := SomeFunction;      // Zuweisung von SomeFunction an F
  I := F(4);             // Funktionsaufruf; Zuweisung des Ergebnisses an I
```

In Zuweisungen bestimmt der Typ der Variablen auf der linken Seite, wie die Prozedur oder Methode auf der rechten Seite interpretiert wird. Beispiel:

```
var
  F, G: function: Integer;
  I: Integer;
  function SomeFunction: Integer;
  ...
  F := SomeFunction;      // Zuweisung von SomeFunction an F
  G := F;                 // Kopieren von F zu G
  I := G;                 // Funktionsaufruf; Zuweisung des Ergebnisses an I
```

Die erste Anweisung weist `F` einen prozeduralen Wert zu. Die zweite Anweisung kopiert diesen Wert in eine andere Variable. In der dritten Anweisung wird die referenzierte Funktion aufgerufen und das Ergebnis `I` zugewiesen. Da `I` keine prozedurale, sondern eine Integer-Variable ist, führt die letzte Zuweisung zum Aufruf der Funktion (die als Ergebnis einen Integer-Wert zurückgibt).

Es gibt Situationen, in denen nicht offensichtlich ist, wie eine prozedurale Variable interpretiert werden muss. Sehen Sie sich dazu folgende Anweisung an:

```
if F = MyFunction then ...;
```

In diesem Fall führt `F` zum Aufruf einer Funktion. Der Compiler ruft zuerst die Funktion auf, die von `F` referenziert wird, und dann die Funktion `MyFunction`. Anschließend werden die Ergebnisse verglichen. Hier gilt folgende Regel: Eine prozedurale Variable innerhalb eines Ausdrucks stellt einen Aufruf der referenzierten Prozedur oder Funktion dar. Referenziert `F` eine Prozedur (die keinen Wert als Ergebnis liefert) oder eine Funktion, an die Parameter übergeben werden müssen, führt die obige Anweisung zu einem Compiler-Fehler. Um den prozeduralen Wert von `F` mit `MyFunction` zu vergleichen, verwenden Sie folgende Anweisung:

```
if @F = @MyFunction then ...;
```

`@F` konvertiert `F` in eine untypisierte Zeigervariable, die eine Adresse enthält, und `@MyFunction` liefert die Adresse von `MyFunction`.

Um anstelle der in einer prozeduralen Variablen gespeicherten Adresse ihre Speicheradresse zu erhalten, verwenden Sie `@@`. So liefert beispielsweise `@@F` die Adresse von `F`.

Der Operator `@` kann auch verwendet werden, um einer prozeduralen Variablen einen untypisierten Zeigerwert zuzuweisen. Beispiel:

```
var StrComp: function(Str1, Str2: PChar): Integer;
  ...
@StrComp := GetProcAddress(KernelHandle, 'lstrcmpi');
```

Diese Anweisung ruft die Funktion `GetProcAddress` auf und weist `StrComp` das Ergebnis zu.

Jede prozedurale Variable kann den Wert `nil` enthalten, was bedeutet, dass sie auf keine bestimmte Adresse zeigt. Der Aufruf einer prozeduralen Variablen, die den Wert `nil` enthält, führt zu einem Fehler. Mit der Standardfunktion `Assigned` können Sie

feststellen, ob einer prozeduralen Variablen ein Wert zugewiesen ist:

```
if Assigned(OnClick) then OnClick(X);
```

Siehe auch

Datentypen ([siehe Seite 887](#))

Einfache Typen ([siehe Seite 889](#))

String-Typen ([siehe Seite 896](#))

Strukturierte Typen ([siehe Seite 902](#))

Zeiger und Zeigertypen ([siehe Seite 911](#))

Variante Typen ([siehe Seite 917](#))

Kompatibilität und Identität von Typen ([siehe Seite 921](#))

Typdeklaration ([siehe Seite 923](#))

Variablen ([siehe Seite 924](#))

Deklarierte Konstanten ([siehe Seite 926](#))

4.1.1.4.7 Variante Typen

In diesem Thema wird die Verwendung von varianten Typen erläutert.

Allgemeines zu Varianten

Es gibt Situationen, in denen Daten verarbeitet werden müssen, deren Typ sich zur Laufzeit dynamisch ändert oder zur Compilierzeit noch nicht bekannt ist. In derartigen Fällen werden Variablen und Parameter des Typs Variant eingesetzt. Varianten bieten zwar eine größere Flexibilität, belegen aber auch mehr Speicher als normale Variablen. Operationen mit Varianten verlaufen deshalb deutlich langsamer als solche mit statischen Typen. Außerdem führen unzulässige Operationen, die bei regulären Variablen während des Compilierens bereinigt werden, bei Varianten oft zu Laufzeitfehlern. Sie können auch benutzerdefinierte variante Typen erstellen.

Eine Variante kann zur Laufzeit die verschiedensten Typen annehmen. Records, Mengen, statische Arrays, Dateien, Klassen, Klassenreferenzen und Zeiger sind jedoch standardmäßig nicht erlaubt. Varianten können also Werte beliebigen Typs mit Ausnahme von strukturierten Typen und Zeigern enthalten. Wenn Varianten Interfaces enthalten, kann über diese auf ihre Methoden und Eigenschaften zugegriffen werden. (Weitere Informationen finden Sie unter Objekt-Interfaces ([siehe Seite 1006](#).) Varianten können auch dynamische Arrays und variante Arrays (ein spezieller Typ von statischen Arrays) aufnehmen (siehe "Variante Arrays"). Varianten können in Ausdrücken und Zuweisungen mit anderen Varianten, aber auch mit Integer-Werten, reellen Werten, Strings und Booleschen Werten kombiniert werden. Die erforderlichen Typumwandlungen nimmt der Compiler automatisch vor.

Varianten, die Strings enthalten, können nicht indiziert werden. Wenn die Variante `V` beispielsweise einen String-Wert enthält, führt die Konstruktion `V[1]` zu einem Laufzeitfehler.

Sie können benutzerdefinierte Varianten definieren, wodurch der Typ Variant beliebige Werte enthalten kann. So können Sie beispielsweise einen Varianten-String-Typ definieren, der die Indizierung ermöglicht oder der bestimmte Klassenverweise, Record-Typen oder statische Arrays enthält. Benutzerdefinierte Variantentypen lassen sich definieren, indem Sie Nachkommen der Klasse `TCustomVariantType` erstellen.

Anmerkung: Diese und fast die gesamte übrige Funktionalität von Varianten wird in der Unit `Variant` implementiert.

Eine Variante belegt 16 Byte Speicher und besteht aus einem Typencode und einem Wert (bzw. einem Zeiger auf einen Wert), dessen Typ durch den Code festgelegt ist. Alle Varianten werden bei der Erstellung mit dem speziellen Wert `Unassigned`

initialisiert. Der Wert `Null` steht für unbekannte oder fehlende Daten.

Der Typencode einer Variante kann mit der Standardfunktion `VarType` ermittelt werden. Die Konstante `varTypeMask` ist eine Bit-Maske, mit der der Typencode aus dem Rückgabewert der Funktion `VarType` isoliert werden kann. Der Ausdruck

`VarType(V) and varTypeMask = varDouble`

ergibt beispielsweise `True`, wenn `V` einen **Double**-Wert oder ein Array solcher Werte enthält. Die Maske verbirgt einfach das erste Bit, das anzeigt, ob die Variante ein Array enthält. Der in der Unit `System` definierte Record-Typ `TVarData` ermöglicht die Typumwandlung einer Variante, um Zugriff auf deren interne Darstellung zu erhalten.

Typkonvertierung bei Varianten

Alle Integer-Typen, reellen Typen, Strings, Zeichentypen und Booleschen Typen sind zum Typ `Variant` zuweisungskompatibel. Ausdrücke können explizit in eine Variante konvertiert werden. Zusätzlich kann die interne Darstellung einer Variante mit den Standardroutinen `VarAsType` und `VarCast` verändert werden. Das folgende Beispiel zeigt die Verwendung von Varianten und die automatische Konvertierung, die bei einer Kombination von Varianten mit anderen Typen durchgeführt wird.

```
var
  V1, V2, V3, V4, V5: Variant;
  I: Integer;
  D: Double;
  S: string;
begin
  V1 := 1; { Integer-Wert }
  V2 := 1234.5678; { Real-Wert }
  V3 := 'Hello world!'; { String-Wert }
  V4 := '1000'; { String-Wert }
  V5 := V1 + V2 + V4; { Real-Wert 2235.5678 }
  I := V1; { I = 1 (Integer-Wert) }
  D := V2; { D = 1234.5678 (Real-Wert) }
  S := V3; { S = 'Hello world!' (String-Wert) }
  I := V4; { I = 1000 (Integer-Wert) }
  S := V5; { S = '2235.5678' (String-Wert) }
end;
```

Die folgende Tabelle enthält die Regeln, die bei der Konvertierung von Varianten in andere Typen gelten.

Regeln für die Typkonvertierung bei Varianten

Zieltyp Ausgangstyp	Integer	Real	String	Boolean
Integer	Konvertiert Integer-Formate	Konvertiert reelle Werte in String-Darstellung		Ergibt False , wenn 0, andernfalls True
Real	Rundet zum nächsten Integer	Konvertiert reelle Formate	Konvertiert String-Darstellung (länder spezifische Einstellungen werden verwendet)	Ergibt False , wenn 0, andernfalls True
String	Konvertiert in Integer (Wert wird evtl. abgeschnitten); wenn der String nicht numerisch ist, wird eine Exception ausgelöst	Konvertiert in reelle Werte (länder spezifische Einstellungen werden verwendet); wenn der String nicht numerisch ist, wird eine Exception ausgelöst	Konvertiert String-Zeichenformate	Ergibt False , wenn der String "false" ist (ohne Berücksichtigung der Groß-/Kleinschreibung) oder ein numerischer String, der zu 0 ausgewertet wird; ergibt True , wenn der String "true" oder ein numerischer String ungleich 0 ist; andernfalls wird eine Exception ausgelöst

Zeichen	Identisch mit String (oben)	Identisch mit String (oben)	Identisch mit String (oben)	Identisch mit String (oben)
Boolean	False = 0, True : alle Bits werden 1 gesetzt (-1 wenn Integer, 255 wenn Byte, etc.)	False = 0, True = 1	False = 'False', True = 'True' per Vorgabe; hängt von der globalen Variable BooleanToStringRule ab	False = False , True = True
Unassigned	Ergibt 0	Ergibt 0	Ergibt einen leeren String	Ergibt False
Null	hängt von der globalen Variable NullStrictConvert ab (löst per Vorgabe eine Exception aus)	hängt von der globalen Variable NullStrictConvert ab (löst per Vorgabe eine Exception aus)	hängt von den globalen Variablen NullStrictConvert und NullAsStringValue ab (löst per Vorgabe eine Exception aus)	hängt von der globalen Variable NullStrictConvert ab (löst per Vorgabe eine Exception aus)

Wenn Zuweisungen außerhalb des zulässigen Bereichs liegen, wird der Zielvariablen meist der höchste Wert im Bereich zugewiesen. Ungültige Variantenoperationen, Zuweisungen oder Umwandlungen führen zu einer `EVariantError`-Exception oder einer Exception-Kasse, die von `EVariantError` abgeleitet ist.

Für den in der Unit `System` deklarierten Typ `TDateTime` gelten gesonderte Konvertierungsregeln. `TDateTime`-Variablen werden bei einer Typumwandlung als normaler **Double**-Wert behandelt. Wenn ein Integer-Wert, ein reeller Wert oder ein Boolescher Wert in den Typ `TDateTime` konvertiert wird, erfolgt zunächst eine Umwandlung in den Typ **Double**. Anschließend wird der Wert als Datums-/Zeitwert gelesen. Bei der Umwandlung eines Strings in den Typ `TDateTime` wird der String unter Verwendung der länderspezifischen Einstellungen als Datums-/Zeitwert interpretiert. Ein in den Typ `TDateTime` konvertierter `Unassigned`-Wert wird als reeller oder als Integer-Wert 0 behandelt. Bei der Konvertierung eines `Null`-Wertes in den Typ `TDateTime` wird eine Exception ausgelöst.

Für eine Variante, die in Win32 ein COM-Interface referenziert, wird beim Versuch einer Konvertierung die Standardeigenschaft des Objekts gelesen und der betreffende Wert in den geforderten Typ umgewandelt. Besitzt das Objekt keine Standardeigenschaft, wird eine Exception ausgelöst.

Varianten in Ausdrücken

Alle Operatoren außer `^`, `is` und `in` unterstützen variante Operanden. Mit Ausnahme von Vergleichen, die stets zu einem Booleschen Ergebnis führen, liefern alle Operationen an varianten Werten ein variantes Ergebnis. Kombiniert ein Ausdruck Varianten mit Werten statischen Typs werden diese Werte automatisch in Varianten konvertiert.

Dies gilt nicht für Vergleiche, bei denen eine beliebige Operation an einer Null-Variante zu einer `Null`-Variante führt. Beispiel:

```
v := Null + 3;
```

Hier wird `v` eine `Null`-Variante zugewiesen. Standardmäßig behandeln Vergleiche eine `Null`-Variante als eindeutigen Wert, der geringer als jeder andere Wert ist. Beispiel:

```
if Null > -3 then ... else ...;
```

In diesem Beispiel wird der `else`-Teil der `if`-Anweisung ausgeführt. Dieses Verhalten lässt sich durch Zuweisen der globalen Varianten `NullEqualityRule` und `NullMagnitudeRule` ändern.

Variante Arrays

Einer Varianten kann kein normales statisches Array zugewiesen werden. Vielmehr wird ein variantes Array benötigt, das durch einen Aufruf der Standardfunktionen `VarArrayCreate` oder `VarArrayOf` erzeugt werden kann. Beispiel:

```
v : Variant;
  ...
v := VarArrayCreate([0,9], varInteger);
```

Diese Anweisung erzeugt ein variantes Array mit 10 Integer-Werten und weist es der Variante `v` zu. Das Array kann mit `v[0]`, `v[1]` usw. indiziert werden. Es ist aber nicht möglich, Elemente eines varianten Arrays als `var`-Parameter zu übergeben. Variante Arrays werden immer mit Integer-Werten indiziert.

Der zweite Parameter in einem Aufruf von `VarArrayCreate` ist der Typencode für den Basistyp des Arrays. Eine Liste dieser Codes finden Sie unter `VarType`. Die Übergabe des Codes `varString` an `VarArrayCreate` ist nicht zulässig. Für variante String-Arrays muss der Typencode `varOleStr` verwendet werden.

Varianten können variante Arrays mit unterschiedlichen Größen, Dimensionen und Basistypen enthalten. Die Elemente eines varianten Arrays können jeden in Varianten zulässigen Typ mit Ausnahme von **ShortString** und **AnsiString** haben. Wenn das Array den Basistyp **Variant** hat, können die Elemente sogar heterogen sein. Die Größe eines varianten Arrays kann mit der Funktion `VarArrayRedim` verändert werden. Weitere Routinen zur Bearbeitung varianter Arrays sind `VarArrayDimCount`, `VarArrayLowBound`, `VarArrayHighBound`, `VarArrayRef`, `VarArrayLock` und `VarArrayUnlock`.

Anmerkung: Variante Arrays benutzerdefinierter Varianten werden nicht unterstützt, da Instanzen benutzerdefinierter Varianten dem varianten Array `VarVariant` zugewiesen werden.

Wenn eine Variante, die ein variantes Array enthält, einer anderen Variante zugewiesen oder als Wertparameter übergeben wird, entsteht eine Kopie des gesamten Arrays. Dieser Vorgang ist jedoch sehr speicherintensiv und sollte möglichst vermieden werden.

OleVariant

Der Typ **OleVariant** wird sowohl auf Windows- als auch auf Linux-Plattformen unterstützt. Der Hauptunterschied zwischen **Variant** und **OleVariant** besteht darin, dass **Variant** Datentypen enthalten kann, die nur der aktuellen Anwendung bekannt sind. **OleVariant** kann nur Datentypen enthalten, die zur OLE-Automatisierung kompatibel sind. Bei diesen Typen ist eine korrekte Verarbeitung der Daten sichergestellt, wenn sie programm- oder netzwerkübergreifend übergeben werden.

Wenn eine **Variante** mit benutzerdefinierten Daten (etwa einem Delphi-String oder einer der neuen benutzerdefinierten varianten Typen) einer **OleVariant**-Variablen zugewiesen wird, versucht die Laufzeitbibliothek, die **Variante** in einen der Standarddatentypen von **OleVariant** umzuwandeln (z. B. einen Delphi-String in einen OLE-BSTR-String). Wenn Sie beispielsweise eine Variante, die einen **AnsiString**-Wert enthält, einer **OleVariant**-Variablen zuweisen, wird der **AnsiString**-Wert in den Typ **WideString** konvertiert. Dasselbe gilt bei der Übergabe einer **Variante** an einen **OleVariant**-Funktionsparameter.

Siehe auch

Datentypen ([siehe Seite 887](#))

Einfache Typen ([siehe Seite 889](#))

String-Typen ([siehe Seite 896](#))

Strukturierte Typen ([siehe Seite 902](#))

Zeiger und Zeigertypen ([siehe Seite 911](#))

Prozedurale Typen ([siehe Seite 914](#))

Kompatibilität und Identität von Typen ([siehe Seite 921](#))

Typdeklaration ([siehe Seite 923](#))

Variablen ([siehe Seite 924](#))

Deklarierte Konstanten ([siehe Seite 926](#))

4.1.1.4.8 Kompatibilität und Identität von Typen

Um zu verstehen, welche Operationen mit unterschiedlichen Ausdrücken durchgeführt werden können, müssen Ihnen die verschiedenen Arten der Kompatibilität zwischen Typen und Werten bekannt sein. Dies beinhaltet Folgendes:

- Typidentität
- Typkompatibilität
- Zuweisungskompatibilität

Typidentität

Wenn ein Typbezeichner mit einem anderen Typbezeichner deklariert wird (ohne Qualifizierer), bezeichnen beide denselben Typ. In den Deklarationen

```
type
  T1 = Integer;
  T2 = T1;
  T3 = Integer;
  T4 = T2;
```

sind T1, T2, T3, T4 und **Integer** identische Typen. Um unterschiedliche Typen zu definieren, wiederholen Sie das Wort **type** in der Deklaration. Beispiel:

```
type TMyInteger = type Integer;
```

Diese Anweisung erzeugt einen neuen Typ namens **TMyInteger**, der nicht mit **Integer** identisch ist.

Sprachkonstrukte, die als Typnamen fungieren, erzeugen bei jedem Auftreten einen anderen Typ. Die Typdeklarationen

```
type
  TS1 = set of Char;
  TS2 = set of Char;
```

definieren beispielsweise die beiden unterschiedlichen Typen TS1 und TS2. Entsprechend werden mit den Variablen Deklarationen

```
var
  S1: string[10];
  S2: string[10];
```

zwei Variablen unterschiedlichen Typs erzeugt. Um Variablen mit identischem Typ zu deklarieren, gehen Sie folgendermaßen vor:

```
var S1, S2: string[10];
```

oder

```
type MyString = string[10];
var
  S1: MyString;
  S2: MyString;
```

Typkompatibilität

Jeder Typ ist mit sich selbst kompatibel. Zwei verschiedene Typen sind kompatibel, wenn sie mindestens eine der folgenden Bedingungen erfüllen:

- Beide Typen sind reelle Typen.
- Beide Typen sind Integer-Typen.
- Ein Typ ist ein Teilbereich des anderen.
- Beide Typen sind Teilbereiche desselben Typs.
- Beide Typen sind Mengentypen mit kompatiblen Basistypen.

- Beide Typen sind gepackte String-Typen mit identischer Anzahl von Zeichen.
- Ein Typ ist ein String-Typ, der andere ist ein String-Typ, ein gepackter String-Typ oder der Typ **Char**.
- Ein Typ ist eine Variante (Typ Variant), der andere ein Integer-Typ, ein reeller Typ, ein String-Typ, ein Zeichentyp oder ein Boolescher Typ.
- Beide Typen sind Klassentypen, Klassenreferenztypen oder Interface-Typen, wobei der eine Typ vom anderen abgeleitet ist.
- Ein Typ ist **PChar** oder **PWideChar**, der andere ein nullbasiertes Zeichen-Array der Form `array[0..n] of PChar` oder **PWideChar**.
- Ein Typ ist vom Typ **Pointer** (untypisierter Zeiger), der andere ist ein beliebiger Zeigertyp.
- Beide Typen sind (typisierte) Zeiger auf denselben Typ, und die Compiler-Direktive `{$T+}` ist aktiviert.
- Beide Typen sind prozedurale Typen mit identischem Ergebnistyp und identischer Parameteranzahl, wobei Parameter an derselben Position identische Typen haben müssen.

Zuweisungskompatibilität

Bei der Zuweisungskompatibilität handelt es sich nicht um ein symmetrisches Verhältnis. Ein Ausdruck des Typs T2 kann einer Variablen des Typs T1 zugewiesen werden, wenn der Ausdruck im Bereich von T1 liegt und mindestens eine der folgenden Bedingungen erfüllt ist:

- T1 und T2 sind identische Typen, und beide sind weder ein Dateityp noch ein strukturierter Typ, der Dateikomponenten enthält.
- T1 und T2 sind kompatible ordinale Typen.
- T1 und T2 sind reelle Typen.
- T1 ist ein reeller Typ, und T2 ist ein Integer-Typ.
- T1 ist ein **PChar**, **PWideChar** oder ein String-Typ, und der Ausdruck ist eine String-Konstante.
- T1 und T2 sind String-Typen.
- T1 ist ein String-Typ, und T2 ist ein gepackter String-Typ oder ein Zeichentyp (**Char**)
- T1 ist ein langer String, und T2 ist ein **PChar** oder **PWideChar**.
- T1 und T2 sind kompatible gepackte String-Typen.
- T1 und T2 sind kompatible Mengentypen.
- T1 und T2 sind kompatible Zeigertypen.
- T1 und T2 sind Klassentypen, Klassenreferenztypen oder Interface-Typen, wobei T2 von T1 abgeleitet ist.
- T1 ist ein Interface-Typ, und T2 ist ein Klassentyp, der T1 implementiert.
- T1 ist vom Typ **PChar** oder **PWideChar**, und T2 ist ein nullbasiertes Zeichen-Array der Form `array[0..n] of Char` (wenn T1 gleich **PChar** oder **WideChar** (wenn T1 gleich **PWideChar**).
- T1 und T2 sind kompatible prozedurale Typen. Ein Funktions- oder Prozedurbezeichner wird in bestimmten Zuweisungsanweisungen als Ausdruck eines prozeduralen Typs behandelt.
- T1 ist eine Variante (Typ Variant), und T2 ist ein Integer-Typ, ein reeller Typ, ein String-Typ, ein Zeichentyp, ein Boolescher Typ, ein Interface-Typ oder ein **OleVariant**-Typ.
- T1 ist ein **OleVariant**-Typ, und T2 ist ein Integer-Typ, ein reeller Typ, ein String-Typ, ein Zeichentyp, ein Boolescher Typ, ein Interface-Typ oder eine Variante (Typ Variant).
- T1 ist ein Integer-Typ, ein reeller Typ, ein String-Typ, ein Zeichentyp oder ein Boolescher Typ, und T2 ist eine Variante oder ein **OleVariant**-Typ.
- T1 ist der Interface-Typ **IUnknown** oder **IDispatch**, und T2 ist ein Variant- oder **OleVariant**-Typ. (Die Variante muss den Typencode `varEmpty`, `varUnknown` oder `varDispatch` haben, wenn T1 **IUnknown** ist bzw. `varEmpty` oder `varDispatch`, wenn T1 **IDispatch** ist.)

Siehe auch

- [Datentypen \(siehe Seite 887\)](#)
- [Einfache Typen \(siehe Seite 889\)](#)
- [String-Typen \(siehe Seite 896\)](#)
- [Strukturierte Typen \(siehe Seite 902\)](#)
- [Zeiger und Zeigertypen \(siehe Seite 911\)](#)
- [Prozedurale Typen \(siehe Seite 914\)](#)
- [Variante Typen \(siehe Seite 917\)](#)
- [Typdeklaration \(siehe Seite 923\)](#)
- [Variablen \(siehe Seite 924\)](#)
- [Deklarierte Konstanten \(siehe Seite 926\)](#)

4.1.1.4.9 Typdeklaration

Dieses Thema enthält eine Beschreibung der Deklarationssyntax für Typen in Delphi.

Syntax für Typdeklarationen

Bei einer Typdeklaration wird ein Bezeichner angegeben, der einen Typ festlegt. Die Syntax für eine Typdeklaration lautet

```
type neuerTyp = Typ
```

Hierbei ist *neuerTyp* ein gültiger Bezeichner. Nach der Typdeklaration

```
type TMyString = string;
```

ist beispielsweise die folgende Variablen-deklaration möglich:

```
var S: TMyString;
```

Die Typdeklaration selbst liegt nicht im Gültigkeitsbereich des Typbezeichners (dies gilt allerdings nicht für Zeigertypen). Aus diesem Grund kann beispielsweise ein Record-Typ definiert werden, der sich selbst rekursiv verwendet.

Wenn Sie einen Typ deklarieren, der mit einem vorhandenen Typ identisch ist, behandelt der Compiler den neuen Typbezeichner als Alias für den alten. In den Deklarationen

```
type TValue = Real;
var
  X: Real;
  Y: TValue;
```

haben *X* und *Y* denselben Typ. Zur Laufzeit ist keine Unterscheidung zwischen *TValue* und **Real** möglich (was normalerweise auch nicht nötig ist). Wenn der neue Typ aber Laufzeit-Typinformationen bereitstellen soll (z. B. zur Bearbeitung von Eigenschaften eines bestimmten Typs in einem Eigenschaftseditor), muss eine Unterscheidung zwischen den Typen möglich sein. Verwenden Sie in solchen Fällen die folgende Syntax:

```
type neuerTyp = type Typ
```

Beispiel:

```
type TValue = type Real;
```

Durch diese Anweisung wird der Compiler veranlasst, einen neuen, eigenen Typ namens *TValue* zu erzeugen.

Bei *var*-Parametern müssen formale und aktuelle Typen identisch sein. Beispiel:

```

type
  TMyType = type Integer
  procedure p(var t:TMyType);
begin
end;

procedure x;
var
  m: TMyType;
  i : Integer;
begin
  p(m); // Funktioniert
  p(i); // Fehler! Formale und aktuelle Typen müssen identisch sein.
end;

```

Anmerkung: Dies gilt nur für `var`-Parameter, nicht für `const`- oder Wert-Parameter.

Siehe auch

Datentypen ([siehe Seite 887](#))

Einfache Typen ([siehe Seite 889](#))

String-Typen ([siehe Seite 896](#))

Strukturierte Typen ([siehe Seite 902](#))

Zeiger und Zeigertypen ([siehe Seite 911](#))

Prozedurale Typen ([siehe Seite 914](#))

Variante Typen ([siehe Seite 917](#))

Kompatibilität und Identität von Typen ([siehe Seite 921](#))

Variablen ([siehe Seite 924](#))

Deklarierte Konstanten ([siehe Seite 926](#))

4.1.1.4.10 Variablen

Variablen sind Bezeichner für Werte, die sich zur Laufzeit ändern können. Eine Variable ist sozusagen ein Name für eine Speicheradresse. Sie kann für Lese- und Schreibzugriffe auf diese Speicheradresse verwendet werden. Variablen dienen als Container für Daten eines bestimmten Typs. Die Typisierung liefert dem Compiler Informationen darüber, wie die Daten zu interpretieren sind.

Variablen Deklarationen

Die grundlegende Syntax für eine Variablen Deklaration lautet:

var Bezeichnerliste: Typ;

Bezeichnerliste ist eine durch Komma getrennte Liste gültiger Bezeichner, und *Typ* ist ein beliebiger gültiger Typ. Zum Beispiel:

`var I: Integer;`

Hier wird die Variable `I` mit dem Typ **Integer** deklariert, während

`var X, Y: Real;`

die beiden Variablen `X` und `Y` vom Typ **Real** deklariert.

Bei aufeinander folgenden Deklarationen braucht das reservierte Wort **var** nicht wiederholt zu werden:

```

var
  X, Y, Z: Double;

```

```
I, J, K: Integer;
Digit: 0..9;
Okay: Boolean;
```

Variablen, die innerhalb von Prozeduren und Funktionen deklariert werden, werden als lokale Variablen bezeichnet. Alle anderen Variablen sind globale Variablen. Einer globalen Variable kann bei der Deklaration mit folgender Syntax ein Anfangswert zugewiesen werden:

var Bezeichner: Typ = Konstante;

Konstante ist ein konstanter Ausdruck des Typs *Typ*. Somit entspricht die Deklaration

```
var I: Integer = 7;
```

folgender Deklaration und Anweisung:

```
var I: Integer;
  ...
I := 7;
```

Lokale Variablen können nicht in ihren Deklarationen initialisiert werden. Aufeinander folgende Variablen Deklarationen (z. B. `var X, Y, Z: Real;`) dürfen weder Initialisierungen enthalten noch Deklarationen von Varianten und Dateitypen.

Wenn Sie eine globale Variable nicht explizit initialisieren, wird sie vom Compiler mit 0 initialisiert. Objektinstanzdaten (Felder) werden auch mit 0 initialisiert. Auf der Win32-Plattform ist der Inhalt von lokalen Variablen so lange undefiniert, bis ein Wert zugewiesen wird. Auf der .NET-Plattform initialisiert CLR alle Variablen, einschließlich der lokalen Variablen, mit 0.

Der bei der Deklaration einer Variablen zugewiesene Speicher wird automatisch freigegeben, wenn die Variable nicht mehr benötigt wird. Lokale Variablen werden freigegeben, wenn die Prozedur oder Funktion beendet wird, in der sie deklariert sind. Weitere Informationen über Variablen und die Verwaltung des Speichers finden Sie unter Speicherverwaltung (siehe Seite 1017).

Absolute Adressen

Sie können eine neue Variable an der Adresse erstellen, an der bereits eine Variable existiert. Geben Sie dazu in der Deklaration der neuen Variable nach dem Namen des Typs die Direktive **absolute** und danach den Namen einer vorhandenen (bereits deklarierten) Variable an. Zum Beispiel:

```
var
  Str: string[32];
  StrLen: Byte absolute Str;
```

Diese Anweisung legt fest, dass die Variable `StrLen` an derselben Adresse wie die Variable `Str` beginnt. Da das erste Byte eines kurzen Strings dessen Länge angibt, ist der Wert von `StrLen` die Länge von `Str`.

Die Initialisierung einer Variablen in einer **absolute**-Deklaration und die Kombination von **absolute** mit anderen Direktiven ist nicht zulässig.

Dynamische Variablen

Durch einen Aufruf der Prozedur `GetMem` oder `New` können Sie dynamische Variablen erzeugen. Diese Variablen werden auf dem Heap zugewiesen und nicht automatisch verwaltet. Die Freigabe des für eine dynamische Variable reservierten Speichers liegt in der Verantwortung des Programmierers. Variablen, die mit `GetMem` erzeugt wurden, müssen mit `FreeMem` freigegeben werden. Variablen, die mit `New` erzeugt wurden, müssen mit `Dispose` freigegeben werden. Weitere Standardroutinen zur Verwaltung von dynamischen Variablen sind `ReallocMem`, `AllocMem`, `Initialize`, `Finalize`, `StrAlloc` und `StrDispose`.

Lange Strings, WideStrings, dynamische Arrays, Varianten und Interfaces werden auf dem Heap zugewiesen. Ihr Speicher wird aber dennoch automatisch verwaltet.

Thread-Variablen

Thread-Variablen (oder Thread-lokale Variablen) finden in Multithread-Anwendungen Verwendung. Eine Thread-Variable

entspricht in etwa einer globalen Variablen, die einzelnen Ausführungs-Threads erhalten jedoch eine eigene, private Kopie der Variablen, auf die andere Threads keinen Zugriff haben. Thread-Variablen werden nicht mit **var**, sondern mit **threadvar** deklariert. Zum Beispiel:

```
threadvar X: Integer;
```

Die Deklaration einer Thread-Variablen

- darf nicht innerhalb einer Prozedur oder Funktion erfolgen.
- darf keine Initialisierungen enthalten.
- darf nicht die Direktive **absolute** enthalten.

Dynamische Variablen, die normalerweise vom Compiler verwaltet werden (lange Strings, Strings des Typs WideString, dynamische Arrays, Varianten und Interfaces), können mit der Direktive **threadvar** deklariert werden. Der Compiler gibt aber den auf dem Heap belegten Speicher (der von den einzelnen Ausführungs-Threads reserviert wird) nicht automatisch frei. Wenn Sie diese Datentypen in Thread-Variablen verwenden, müssen Sie selbst für die Freigabe des Speichers innerhalb des Threads sorgen, bevor dieser beendet wird. Zum Beispiel:

```
threadvar S: AnsiString;
S := 'ABCDEFGHIJKLMNPQRSTUVWXYZ';
...
S := ''; // Freigabe des von S belegten Speichers
```

Anmerkung: Die Verwendung solcher Konstrukte wird nicht empfohlen.

Sie können Varianten freigeben, indem Sie ihnen den Status **Unassigned** zuweisen. Um Interfaces oder dynamische Arrays freizugeben, weisen Sie ihnen den Wert **nil** zu.

Siehe auch

Datentypen ([siehe Seite 887](#))

Einfache Typen ([siehe Seite 889](#))

String-Typen ([siehe Seite 896](#))

Strukturierte Typen ([siehe Seite 902](#))

Zeiger und Zeigertypen ([siehe Seite 911](#))

Prozedurale Typen ([siehe Seite 914](#))

Variante Typen ([siehe Seite 917](#))

Kompatibilität und Identität von Typen ([siehe Seite 921](#))

Typdeklaration ([siehe Seite 923](#))

Deklarierte Konstanten ([siehe Seite 926](#))

4.1.1.4.11 Deklarierte Konstanten

Unter dem Begriff Konstanten werden verschiedene Sprachkonstrukte zusammengefasst. Es gibt numerische Konstanten wie die Zahl 17 und String-Konstanten wie "Hello world!". Numerische Konstanten werden auch einfach als Zahlen bezeichnet, String-Konstanten als Zeichen-Strings oder String-Literale. Jeder Aufzählungstyp definiert Konstanten, die den Werten des jeweiligen Typs entsprechen. Es gibt vordefinierte Konstanten wie **True**, **False** und **nil**. Es gibt aber auch Konstanten, die wie Variablen durch eine Deklaration erzeugt werden.

Deklarierte Konstanten sind entweder *echte Konstanten* oder *typisierte Konstanten*. Oberflächlich betrachtet besteht zwischen diesen beiden Konstantentypen kein großer Unterschied. Für ihre Deklaration gelten aber verschiedene Regeln, und sie werden

für unterschiedliche Zwecke eingesetzt.

Echte Konstanten

Unter einer echten Konstante versteht man einen deklarierten Bezeichner, dessen Wert sich nicht ändern kann. Beispiel:

```
const MaxValue = 237;
```

Hier wird die Konstante `MaxValue` mit dem Integer-Wert 237 deklariert. Die Syntax für die Deklaration echter Konstanten lautet:

```
const Bezeichner = konstanterAusdruck
```

Dabei ist *Bezeichner* ein gültiger Bezeichner, und *konstanterAusdruck* ein Ausdruck, der vom Compiler ohne Ausführung des Programms ausgewertet werden kann.

Wenn *konstanterAusdruck* einen ordinalen Wert zurückgibt, kann der Typ der deklarierten Konstante über eine Wertumwandlung festgelegt werden. Beispiel:

```
const MyNumber = Int64(17);
```

Diese Anweisung deklariert die Konstante `MyNumber` mit dem Typ `Int64`, die den Integer-Wert 17 zurückgibt. Wenn der Typ nicht auf diese Weise festgelegt wird, erhält die deklarierte Konstante den Typ von *konstanterAusdruck*.

- Wenn es sich bei *konstanterAusdruck* um einen Zeichen-String handelt, ist die deklarierte Konstante mit jedem String-Typ kompatibel. Hat der Zeichen-String die Länge 1, ist er zusätzlich mit jedem Zeichentyp kompatibel.
- Wenn *konstanterAusdruck* vom Typ `Real` ist, hat er den Typ `Extended`. Ist *konstanterAusdruck* ein Integer, hat er einen der in der folgenden Tabelle aufgeführten Typen.

Typen für Integerkonstanten

Bereich der Konstante (hexadezimal)	Bereich der Konstante (dezimal)	Typ
-\$8000000000000000..-\$80000001	-2^63..-2147483649	Int64
-\$80000000..-\$8001	-2147483648..-32769	Integer
-\$8000..-\$81	-32768..-129	Smallint
-\$80..-1	-128..-1	Shortint
0..\$7F	0..127	0..127
\$80..\$FF	128..255	Byte
\$0100..\$7FFF	256..32767	0..32767
\$8000..\$FFFF	32768..65535	Word
\$10000..\$7FFFFFFF	65536..2147483647	0..2147483647
\$80000000..\$FFFFFFFF	2147483648..4294967295	Cardinal
\$10000000..\$7FFFFFFFFFFFFF	4294967296..2^631	Int64

Hier einige Beispiele für Konstantendeklarationen:

```
const
  Min = 0;
  Max = 100;
  Center = (Max - Min) div 2;
  Beta = Chr(225);
  NumChars = Ord('Z') - Ord('A') + 1;
  Message = 'Zu wenig Speicher';
  ErrStr = ' Error: ' + Message + '. ';
  ErrPos = 80 - Length(ErrStr) div 2;
  Ln10 = 2.302585092994045684;
  Ln10R = 1 / Ln10;
  Numeric = ['0'..'9'];
```

```
Alpha = ['A'..'Z', 'a'..'z'];
AlphaNum = Alpha + Numeric;
```

Konstante Ausdrücke

Ein konstanter Ausdruck kann vom Compiler ohne Ausführung des Programms ausgewertet werden. Als konstante Ausdrücke können Zahlen, Zeichen-Strings, echte Konstanten, Werte von Aufzählungstypen und die speziellen Konstanten **True**, **False** und **nil** verwendet werden. Zulässig sind auch Ausdrücke, die sich aus diesen Elementen zusammensetzen und Operatoren, Typumwandlungen und Mengenkonstruktoren enthalten. In konstanten Ausdrücken können keine Variablen, Zeiger und Funktionsaufrufe verwendet werden. Eine Ausnahme bilden Aufrufe der folgenden, vordefinierten Funktionen:

Abs	High	Low	Pred	Succ
Chr	Length	Odd	Round	Swap
Hi	Lo	Ord	SizeOf	Trunc

Die Definition eines konstanten Ausdrucks wird in verschiedenen Bereichen der Syntaxdefinition von Delphi verwendet. Konstante Ausdrücke werden zur Initialisierung globaler Variablen, zur Definition von Teilbereichstypen, zur Zuweisung von Ordinalpositionen an Werte in Aufzählungstypen, zur Festlegung von Standardparameterwerten, zur Erstellung von **case**-Anweisungen und zur Deklaration von echten und typisierten Konstanten eingesetzt.

Hier einige Beispiele für konstante Ausdrücke:

```
100
'A'
256 - 1
(2.5 + 1) / (2.5 - 1)
'Borland' + ' ' + 'Developer'
Chr(32)
Ord('Z') - Ord('A') + 1
```

Ressourcenstrings

Ressourcen-Strings werden als Ressource gespeichert und zu einer ausführbaren Datei oder Bibliothek gelinkt. Deshalb ist keine erneute Compilierung des Programms erforderlich, wenn ein Ressourcen-String geändert wird.

Ressourcen-Strings werden wie echte Konstanten deklariert, wobei das Wort **const** durch **resourcestring** ersetzt wird. Der Ausdruck rechts neben dem Symbol **=** muss ein konstanter Ausdruck sein, der als Ergebnis einen String-Wert liefert. Beispiel:

```
resourcestring
  CreateError = 'Datei %s kann nicht erstellt werden';
  OpenError = 'Datei %s kann nicht geöffnet werden';
  LineTooLong = 'Zeile zu lang';
  ProductName = 'CodeGear Rocks';
  SomeResourceString = SomeTrueConstant;
```

Typisierte Konstanten

Typisierte Konstanten können im Gegensatz zu echten Konstanten auch Werte mit Array-, Record-, Zeiger- und prozedurellem Typ enthalten. Konstante Ausdrücke dürfen keine typisierten Konstanten enthalten.

Typisierte Konstanten werden folgendermaßen deklariert:

const Bezeichner: Typ = Wert

Dabei ist **Bezeichner** ein gültiger Bezeichner, **Typ** ist jeder beliebige Typ mit Ausnahme von Dateitypen und Varianten, und **Wert** ist ein Ausdruck des Typs **Typ**. Beispiel:

```
const Max: Integer = 100;
```

In den meisten Fällen muss *Wert* ein konstanter Ausdruck sein. Wenn für *Typ* ein Array-, Record-, Zeiger- oder prozeduraler Typ angegeben wird, gelten spezielle Regeln.

Array-Konstanten

Bei der Deklaration einer Array-Konstante werden die Werte der Array-Elemente in Klammern und durch Kommas getrennt angegeben. Zur Angabe der Werte müssen konstante Ausdrücke verwendet werden. Beispiel:

```
const Digits: array[0..9] of Char = ('0', '1', '2', '3', '4', '5', '6', '7', '8', '9');
```

Diese Anweisung deklariert eine typisierte Konstante mit dem Namen `Digits`, die ein Zeichen-Array enthält.

Da nullbasierte Zeichen-Arrays oft nullterminierte Strings darstellen, können zur Initialisierung von Zeichen-Arrays Konstanten verwendet werden. Die obige Deklaration lässt sich folgendermaßen vereinfachen:

```
const Digits: array[0..9] of Char = '0123456789';
```

Konstanten, die mehrdimensionale Arrays darstellen, werden deklariert, indem die Konstanten jeder einzelnen Dimension in Klammern gesetzt und durch Kommas voneinander getrennt werden. Beispiel:

```
type TCube = array[0..1, 0..1, 0..1] of Integer;
const Maze: TCube = (((0, 1), (2, 3)), ((4, 5), (6, 7)));
```

Hier wird ein Array namens `Maze` mit den folgenden Werten erstellt:

```
Maze[0,0,0] = 0
Maze[0,0,1] = 1
Maze[0,1,0] = 2
Maze[0,1,1] = 3
Maze[1,0,0] = 4
Maze[1,0,1] = 5
Maze[1,1,0] = 6
Maze[1,1,1] = 7
```

Array-Konstanten dürfen auf keiner Ebene Dateitypen enthalten.

Record-Konstanten

Bei der Deklaration einer Record-Konstante geben Sie am Ende der Deklaration den Wert jedes Feldes in Klammern an. Die Angabe erfolgt in der Form *Feldname*: *Wert*, wobei die einzelnen Feldzuweisungen durch einen Strichpunkt voneinander getrennt werden. Die Werte müssen durch konstante Ausdrücke dargestellt werden. Die Reihenfolge der Felder muss der Reihenfolge bei der Deklaration des Record-Typs entsprechen. Wenn ein Tag-Feld vorhanden ist, muss dessen Wert angegeben werden. Enthält der Record einen varianten Teil, können Werte nur an die Variante zugewiesen werden, die durch das Tag-Feld angegeben ist.

Beispiele:

```
type
  TPoint = record
    X, Y: Single;
  end;
  TVector = array[0..1] of TPoint;
  TMonth = (Jan, Feb, Mar, Apr, Mai, Jun, Jul, Aug, Sep, Okt, Nov, Dez);
  TDate = record
    D: 1..31;
    M: TMonth;
    Y: 1900..1999;
  end;
const
  Origin: TPoint = (X: 0,0; Y: 0,0);
  Line: TVector = ((X: -3,1; Y: 1,5), (X: 5,8; Y: 3,0));
  SomeDay: TDate = (D: 2; M: Dez; Y: 1960);
```

Record-Konstanten dürfen auf keiner Ebene Dateitypen enthalten.

Prozedurale Konstanten

Zur Deklaration einer prozeduralen Konstante geben Sie den Namen einer Funktion oder Prozedur an, die mit dem deklarierten Typ der Konstante kompatibel ist. Beispiel:

```
function Calc(X, Y: Integer): Integer;
begin
  ...
end;

type TFunction = function(X, Y: Integer): Integer;
const MyFunction: TFunction = Calc;
```

Die prozedurale Konstante `MyFunction` kann dann in einem Funktionsaufruf eingesetzt werden:

```
I := MyFunction(5, 7)
```

Prozeduralen Konstanten kann auch der Wert `nil` zugewiesen werden.

Zeigerkonstanten

Bei der Deklaration einer Zeigerkonstante muss diese mit einem Wert initialisiert werden, der vom Compiler (zumindest als relative Adresse) aufgelöst werden kann. Dies ist auf drei Arten möglich: mit dem Operator `@`, mit `nil` oder (bei **PChar**- oder **PWideChar**-Konstanten) mit einem String-Literal. Ist z. B. `I` eine globale Variable des Typs **Integer**, können Sie folgende Konstante deklarieren:

```
const PI: ^Integer = @I;
```

Die Konstante kann vom Compiler aufgelöst werden, da globale Variablen Bestandteil des Codesegments sind. Dasselbe gilt für Funktionen und globale Konstanten:

```
const PF: Pointer = @MyFunction;
```

Da String-Literale als globale Konstanten zugewiesen werden, können Sie eine **PChar**-Konstante mit einem String-Literal initialisieren:

```
const WarningStr: PChar = 'Warnung!';
```

Siehe auch

[Datentypen](#) (siehe Seite 887)

[Einfache Typen](#) (siehe Seite 889)

[String-Typen](#) (siehe Seite 896)

[Strukturierte Typen](#) (siehe Seite 902)

[Zeiger und Zeigertypen](#) (siehe Seite 911)

[Prozedurale Typen](#) (siehe Seite 914)

[Variante Typen](#) (siehe Seite 917)

[Kompatibilität und Identität von Typen](#) (siehe Seite 921)

[Typdeklaration](#) (siehe Seite 923)

[Variablen](#) (siehe Seite 924)

4.1.1.5 Prozeduren und Funktionen

In diesem Abschnitt wird die Syntax von Funktions- und Prozedurdeklarationen beschrieben.

4.1.1.5.1 Prozeduren und Funktionen

Dieses Thema enthält Informationen zu folgenden Bereichen:

- Prozeduren und Funktionen deklarieren
- Aufrufkonventionen
- forward- und interface-Deklarationen
- external-Deklarationen
- Prozeduren und Funktionen überladen
- Lokale Deklarationen und verschachtelte Routines

Allgemeines zu Prozeduren und Funktionen

Prozeduren und Funktionen, die zusammenfassend auch als *Routines* bezeichnet werden, sind abgeschlossene Anweisungsblöcke, die von unterschiedlichen Positionen in einem Programm aufgerufen werden können. Eine *Funktion* ist eine Routine, die nach Ausführung einen Wert zurückgibt. Eine *Prozedur* ist eine Routine, die keinen Wert zurückgibt.

Funktionsaufrufe können auch in Zuweisungen und Operationen eingesetzt werden, da sie einen Wert zurückgeben. Zum Beispiel:

```
I := SomeFunction(X);
```

Diese Anweisung ruft die Funktion `SomeFunction` auf und weist das Ergebnis der Variablen `I` zu. Funktionsaufrufe dürfen nicht auf der linken Seite einer Zuweisungsanweisung stehen.

Prozedurauftrufe können als eigenständige Anweisungen eingesetzt werden. Bei Funktionsaufrufen ist dies ebenfalls möglich, wenn die erweiterte Syntax `{$X+}` aktiviert ist. Zum Beispiel:

```
DoSomething;
```

Diese Anweisung ruft die Routine `DoSomething` auf. Ist `DoSomething` eine Funktion, wird der Rückgabewert verworfen.

Prozeduren und Funktionen können auch rekursiv aufgerufen werden.

Prozeduren und Funktionen deklarieren

Beim Deklarieren einer Prozedur oder Funktion geben Sie den Namen, die Anzahl und den Typ der Parameter und - wenn es sich um eine Funktion handelt - den Typ des Rückgabewertes an. Dieser Teil der Deklaration wird auch Prototyp, Einleitung oder Kopf genannt. Anschließend schreiben Sie den Code, der beim Aufrufen der Prozedur oder Funktion ausgeführt werden soll. Dieser Teil wird auch als Rumpf oder Block der Routine bezeichnet.

Prozedurdeklarationen

Eine Prozedurdeklaration hat folgende Form:

```
procedure procedureName(parameterList); directives;
  localDeclarations;
begin
  Anweisungen
end;
```

Prozedurname ist ein beliebiger gültiger Bezeichner, *Anweisungen* eine Folge von Anweisungen, die beim Aufruf der Prozedur ausgeführt werden. (*Parameterliste*), *Direktiven* und *LokaleDeklarationen* sind optional.

Ein Beispiel für eine Prozedurdeklaration:

```
procedure NumString(N: Integer; var S: string);
var
  V : Integer;
begin
  V := Abs(N);
```

```

S := '';
repeat
  S := Chr(V mod 10 + Ord('0')) + S;
  V := V div 10;
until V = 0;
if N < 0 then S := '-' + S;
end;

```

Nach dieser Deklaration können Sie die Prozedur `NumString` folgendermaßen aufrufen:

```
NumString(17, MyString);
```

Dieser Prozederaufruf weist der Variablen `MyString` (es muss sich um eine Variable vom Typ **String** handeln) den Wert 17 zu.

Im Anweisungsblock einer Prozedur können Sie Variablen und Bezeichner verwenden, die im Abschnitt *LokaleDeklarationen* der Prozedur deklariert wurden. Sie können außerdem die Parameternamen aus der Parameterliste (`N` und `S` in diesem Beispiel) verwenden. Die Parameterliste definiert eine Gruppe lokaler Variablen. Aus diesem Grund dürfen im Abschnitt *LokaleDeklarationen* keine gleichlautenden Parameternamen auftauchen. Natürlich können Sie auch alle Bezeichner verwenden, in deren Gültigkeitsbereich die Prozedur deklariert wurde.

Funktionsdeklarationen

Eine Funktionsdeklaration entspricht einer Prozedurdeklaration, definiert aber zusätzlich einen Rückgabetyp und einen Rückgabewert. Funktionsdeklarationen haben folgende Form:

```

function Funktionsname(Parameterliste): returnType; directives;
  localDeclarations;
begin
  Anweisungen
end;

```

Funktionsname ist ein beliebiger gültiger Bezeichner, *Rückgabetyp* ein Typbezeichner, *Anweisungen* enthält die Folge von Anweisungen, die beim Aufruf der Funktion ausgeführt werden sollen. (*Parameterliste*), *Direktiven* und *LokaleDeklarationen* sind optional.

Für den Anweisungsblock der Funktion gelten die Regeln, die bereits für Prozeduren erläutert wurden. Sie können Variablen und Bezeichner verwenden, die im Abschnitt *LokaleDeklarationen* der Funktion deklariert wurden. Sie können außerdem die Parameternamen aus der Parameterliste verwenden. Natürlich können Sie auch alle Bezeichner verwenden, in deren Gültigkeitsbereich die Funktion deklariert wurde. Der Funktionsname ist eine spezielle Variable, die wie die vordefinierte Variable **Result** den Rückgabewert der Funktion enthält.

Wenn die erweiterte Syntax `({$X+})` aktiviert ist, wird **Result** implizit in jeder Funktion deklariert. Sie dürfen diese Variable deshalb nicht manuell deklarieren.

Zum Beispiel:

```

function WF: Integer;
begin
  WF := 17;
end;

```

Diese Deklaration definiert eine Konstantenfunktion namens `WF`, die keine Parameter entgegennimmt und immer den Integerwert 17 zurückgibt. Diese Deklaration ist zur folgenden äquivalent:

```

function WF: Integer;
begin
  Result := 17;
end;

```

Das nächste Beispiel enthält eine etwas komplexere Funktionsdeklaration:

```

function Max(A: array of Real; N: Integer): Real;
var
  X: Real;
  I: Integer;

```

```

begin
  X := A[0];
  for I := 1 to N - 1 do
    if X < A[I] then X := A[I];
  Max := X;
end;

```

Sie können der Variablen **Result** oder dem Funktionsnamen im Anweisungsblock mehrmals einen Wert zuweisen. Die zugewiesenen Werte müssen jedoch dem deklarierten Rückgabetyp entsprechen. Sobald die Ausführung der Funktion beendet wird, bildet der Wert, der zuletzt der Variablen **Result** oder dem Funktionsnamen zugewiesen wurde, den Rückgabewert der Funktion. Zum Beispiel:

```

function Power(X: Real; Y: Integer): Real;
var
  I: Integer;
begin
  Result := 1,0;
  I := Y;
  while I > 0 do
  begin
    if Odd(I) then Result := Result * X;
    I := I div 2;
    X := Sqr(X);
  end;
end;

```

Result und der Funktionsname repräsentieren immer denselben Wert. Deshalb gibt die Deklaration

```

function MyFunction: Integer;
begin
  MyFunction := 5;
  Result := Result * 2;
  MyFunction := Result + 1;
end;

```

den Wert 11 zurück. **Result** ist jedoch nicht mit dem Funktionsnamen identisch. Wenn der Funktionsname auf der linken Seite einer Zuweisungsanweisung verwendet wird, verwaltet der Compiler den Funktionsnamen als Variable (wie **Result**) zur Aufzeichnung des Rückgabewerts. Taucht der Funktionsname dagegen an einer anderen Stelle im Anweisungsblock auf, wird er vom Compiler als rekursiver Aufruf der Funktion interpretiert. **Result** kann dagegen als Variable in Operationen eingesetzt werden, beispielsweise bei Typkonvertierungen, in Konstruktoren, Indizes und in Aufrufen anderer Routinen.

Wenn die Ausführung einer Funktion beendet wird, bevor **Result** oder dem Funktionsnamen ein Wert zugewiesen wurde, ist der Rückgabewert der Funktion nicht definiert.

Aufrufkonventionen

Wenn Sie eine Prozedur oder Funktion deklarieren, können Sie eine Aufrufkonvention mit einer der Direktiven **register**, **pascal**, **cdecl**, **stdcall** und **safecall** angeben. Zum Beispiel:

```
function MyFunction(X, Y: Real): Real; cdecl;
```

Aufrufkonventionen bestimmen die Reihenfolge, in der Parameter an die Routine übergeben werden. Sie beeinflussen außerdem das Entfernen der Parameter vom Stack, den Einsatz der Register zur Übergabe von Parametern sowie die Fehler- und Exception-Behandlung. Die Standard-Aufrufkonvention ist **register**.

- Die Konventionen **register** und **pascal** übergeben Parameter von links nach rechts. Der links stehende Parameter wird also zuerst ausgewertet und übergeben, der rechts stehende Parameter zuletzt. Die Konventionen **cdecl**, **stdcall** und **safecall** übergeben die Parameter von rechts nach links.
- Bei allen Konventionen mit Ausnahme von **cdecl** entfernt die Prozedur bzw. Funktion die Parameter vom Stack, sobald die Steuerung zurückgegeben wird. Bei der Konvention **cdecl** entfernt die aufrufende Routine die Parameter vom Stack, sobald sie wieder die Steuerung erhält.
- Die Konvention **register** verwendet bis zu drei CPU-Register zur Übergabe der Parameter, alle anderen Konventionen übergeben die Parameter an den Stack.

- Die Konvention **safecall** implementiert "Exception-Firewalls" und in Win32 die prozessübergreifende COM-Fehlerbenachrichtigung.

Die folgende Tabelle beschreibt die Aufrufkonventionen.

Aufrufkonventionen

Direktive	Parameterreihenfolge	Bereinigung	Parameterübergabe in Registern?
register	Von links nach rechts	Routine	Ja
pascal	Von links nach rechts	Routine	Nein
cdecl	Von rechts nach links	Aufrufender	Nein
stdcall	Von rechts nach links	Routine	Nein
safecall	Von rechts nach links	Routine	Nein

Die weitaus effizienteste Aufrufkonvention ist **register**, da hier meistens kein Stack-Frame für die Parameter angelegt werden muss. (Zugriffsmethoden für Eigenschaften, die als **published** deklariert sind, müssen **register** verwenden.) Die Konvention **cdecl** ist hilfreich, wenn Funktionen in gemeinsam benutzten Bibliotheken aufgerufen werden, die in C oder C++ geschrieben wurden. Die Konventionen **stdcall** und **safecall** sollten für Aufrufe von externem Quelltext benutzt werden. In Win32 verwenden die Betriebssystem-APIs die Konventionen **stdcall** und **safecall**. Andere Betriebssysteme verwenden normalerweise die Konvention **cdecl** (beachten Sie, dass **stdcall** effizienter ist als **cdecl**).

Die Konvention **safecall** muss bei der Deklaration von Methoden für duale Interfaces verwendet werden. Die Konvention **pascal** gewährleistet die Abwärtskompatibilität.

Die Direktiven **near**, **far** und **export** beziehen sich auf die Aufrufkonventionen bei der Programmierung für 16-Bit-Windows-Umgebungen. Sie dienen ausschließlich der Abwärtskompatibilität und haben in Win32- und .NET-Anwendungen keine Auswirkung.

forward- und interface-Deklarationen

Die Direktive **forward** in einer Prozedur- oder Funktionsdeklaration wird durch einen Rumpf mit Anweisungen und lokalen Variablen Deklarationen ersetzt. Zum Beispiel:

```
function Calculate(X, Y: Integer): Real; forward;
```

Die hier deklarierte Funktion Calculate enthält die Direktive **forward**. An einer anderen Position muss der Rumpf der Routine deklariert werden. Diese definierende Deklaration für Calculate kann beispielsweise folgendermaßen aussehen:

```
function Calculate;
  ... {Deklarationen}
begin
  ... {Anweisungsblock}
end;
```

Grundsätzlich braucht eine definierende Deklaration die Parameterliste oder den Rückgabetyp der Routine nicht zu wiederholen. Werden diese Angaben jedoch wiederholt, müssen sie denen in der **forward**-Deklaration exakt entsprechen. Standardparameter in der definierenden Deklaration werden allerdings ignoriert. Wenn die **forward**-Deklaration eine überladene Prozedur oder Funktion definiert, muss die Parameterliste in der definierenden Deklaration wiederholt werden.

Eine **forward**-Deklaration und die dazugehörige definierende Deklaration müssen in demselben Typdeklarationsabschnitt enthalten sein. Sie können also keinen neuen Abschnitt (z. B. einen **var**- oder **const**-Abschnitt) zwischen der **forward**-Deklaration und der definierenden Deklaration einfügen. Die definierende Deklaration kann vom Typ **external** oder **assembler** sein, es darf sich jedoch nicht um eine weitere **forward**-Deklaration handeln.

Mit einer **forward**-Deklaration kann der Gültigkeitsbereich eines Prozedur- oder Funktionsbezeichners auf einen früheren Punkt im Quelltext ausgedehnt werden. Andere Prozeduren und Funktionen können also die mit **forward** deklarierte Routine aufrufen,

bevor sie tatsächlich definiert wurde. **forward**-Deklarationen bieten also nicht nur mehr Flexibilität bei der Programmierung, sie werden auch gelegentlich für Rekursionen benötigt.

Die Direktive **forward** hat keine Auswirkung im **interface**-Abschnitt einer Unit. Prozedur- und Funktions-Header im **interface**-Abschnitt verhalten sich wie **forward**-Deklarationen. Die zugehörigen definierenden Deklarationen müssen in den **implementation**-Abschnitt eingefügt werden. Eine im **interface**-Abschnitt deklarierte Routine ist von jeder Position in der Unit und für jede Unit und jedes Programm verfügbar, die bzw. das die Unit mit der Deklaration einbindet.

external-Deklarationen

Die Direktive **external** ersetzt den Anweisungsblock in einer Prozedur- oder Funktionsdeklaration. Sie ermöglicht das Aufrufen von Routinen, die unabhängig vom aktuellen Programm kompiliert wurden. Externe Routinen stammen aus Objektdateien oder aus dynamisch ladbaren Bibliotheken.

Verwenden Sie zum Importieren einer C-Funktion, die eine variable Anzahl von Parametern übernimmt, die Direktive **varargs**. Zum Beispiel:

```
function printf(Format: PChar): Integer; cdecl; varargs;
```

Die Direktive **varargs** kann nur bei externen Routinen und zusammen mit der Aufrufkonvention **cdecl** eingesetzt werden.

Objektdateien linken

Um Routinen aus einer separat kompilierten Objektdatei aufrufen zu können, müssen Sie die Objektdatei mit der Compiler-Direktive **\$L** (oder **\$LINK**) zur gewünschten Anwendung linken. Zum Beispiel:

```
{$L BLOCK.OBJ}
```

Diese Anweisung linkt die Datei **BLOCK.OBJ** zu dem Programm bzw. der Unit, in der die Anweisung verwendet wird. Anschließend müssen Sie die aufzurufenden Funktionen und Prozeduren deklarieren:

```
procedure MoveWord(var Source, Dest; Count: Integer); external;
procedure FillWord(var Dest; Data: Integer; Count: Integer); external;
```

Jetzt können Sie die Routinen **MoveWord** und **FillWord** in **BLOCK.OBJ** aufrufen.

In Win32 werden Deklarationen wie die oben gezeigten häufig verwendet, um auf externe Assembler-Routinen zuzugreifen. Assembler-Routinen können auch direkt in Delphi-Quelltext eingefügt werden.

Funktionen aus Bibliotheken importieren

Mit einer Direktive der Form

```
external Stringkonstante;
```

können Sie Routinen aus einer dynamisch ladbaren Bibliothek (.DLL-Datei) importieren. Die Anweisung muss an das Ende des Prozedur- bzw. Funktions-Headers angefügt werden. *Stringkonstante* gibt die Bibliotheksdatei in einfachen Anführungszeichen (Hochkommas) an. Ein Beispiel:

```
function SomeFunction(S: string): string; external 'strlib.dll';
```

Mit dieser Anweisung wird in Win32 die Funktion **SomeFunction** aus der Datei **strlib.dll** importiert.

Sie können Routinen unter einem anderen als dem in der Bibliothek verwendeten Namen importieren, indem Sie den Originalnamen in der **external**-Direktive angeben:

```
external Stringkonstante1 name Stringkonstante2;
```

Die erste *Stringkonstante* gibt den Namen der Bibliotheksdatei, die zweite den Originalnamen der Routine an.

Die folgende Deklaration importiert eine Funktion aus **user32.dll** (Teil der Win32-API):

```
function MessageBox(HWND: Integer; Text, Caption: PChar; Flags: Integer): Integer; stdcall;
external 'user32.dll' name 'MessageBoxA';
```

Der Originalname der Funktion lautet `MessageBoxA`, sie wird jedoch unter dem Namen `MessageBox` importiert.

Sie können die zu importierende Routine auch über eine Nummer angeben:

external Stringkonstante index Integerkonstante;

Integerkonstante ist der Index der Routine in der Exporttabelle.

In der Importdeklaration müssen Sie die genaue Schreibweise des Routineenamens verwenden (einschließlich Groß-/Kleinschreibung). Beim Aufruf der importierten Routine wird die Groß-/Kleinschreibung nicht mehr berücksichtigt.

Prozeduren und Funktionen überladen

Sie können mehrere Routinen mit identischen Namen in demselben Gültigkeitsbereich deklarieren. Dieses Verfahren wird *Überladen* genannt. Überladene Routinen müssen mit der Direktive **overload** deklariert werden und unterschiedliche Parameterlisten haben. Beispiele:

```
function Divide(X, Y: Real): Real; overload;
begin
  Result := X/Y;
end

function Divide(X, Y: Integer): Integer; overload;
begin
  Result := X div Y;
end;
```

Diese Deklarationen definieren zwei Funktionen namens `Divide`, die Parameter unterschiedlicher Typen entgegennehmen. Wenn Sie `Divide` aufrufen, ermittelt der Compiler die zu verwendende Funktion durch Prüfung des übergebenen Parametertyps. `Divide(6.0, 3.0)` ruft beispielsweise die erste `Divide`-Funktion auf, da es sich bei den Argumenten um reelle Zahlen handelt.

Einer überladenen Routine können Parameter übergeben werden, deren Typ keinem der in den Routinendeklarationen verwendeten Typen entspricht. Voraussetzung dafür ist, dass diese Parameter zuweisungskompatibel mit den Parametern in mehr als einer Deklaration sind. Dieses Vorgehen wird meist beim Überladen von Routinen mit unterschiedlichen Integer- oder Real-Typen verwendet:

```
procedure Store(X: Longint); overload;
procedure Store(X: Shortint); overload;
```

Wenn die Eindeutigkeit gewährleistet ist, ruft der Compiler in diesen Fällen die Routine auf, deren Parametertyp für den Wertebereich der tatsächlich übergebenen Parameter mindestens erforderlich ist. (Beachten Sie, dass konstante Ausdrücke, die reelle Zahlen sind, immer vom Typ **Extended** sind.)

Überladene Routinen müssen hinsichtlich der Anzahl der entgegengenommenen Parameter oder der Typen dieser Parameter eindeutig sein. Die folgenden beiden Deklarationen führen deshalb zu einem Fehler bei der Compilierung:

```
function Cap(S: string): string; overload;
...
procedure Cap(var Str: string); overload;
...
```

Dagegen sind die folgenden Deklarationen zulässig:

```
function Func(X: Real; Y: Integer): Real; overload;
...
function Func(X: Integer; Y: Real): Real; overload;
...
```

Wird eine überladene Routine mit **forward** oder **interface** deklariert, muss die Parameterliste in der definierenden Deklaration der Routine wiederholt werden.

Der Compiler kann zwischen überladenen Funktionen unterscheiden, die **AnsiString**-/PChar- und

WideString/WideChar-Parameter in derselben Parameterposition enthalten. String-Konstanten oder String-Litale, die in eine solche überladenen Position übergeben werden, werden in den nativen String- oder Zeichentyp übergeben, der **AnsiString/PChar** lautet.

```
procedure test(const S: String); overload;
procedure test(const W: WideString); overload;
var
  a: string;
  b: WideString;
begin
  a := 'a';
  b := 'b';
  test(a);      // Aufruf String-Version
  test(b);      // Aufruf WideString-Version
  test('abc');  // Aufruf String-Version
  test(WideString('abc')); // Aufruf WideString-Version
end;
```

Varianten können ebenfalls als Parameter in überladenen Funktionsdeklarationen verwendet werden. Eine Variante wird allgemeiner angesehen als ein beliebiger einfacher Typ. Exakte Typübereinstimmungen werden stets gegenüber Variantenübereinstimmungen bevorzugt. Wird eine Variante an eine solche überladene Position übergeben, und ist eine **overload**-Anweisung an dieser Parameterposition vorhanden, die eine Variante akzeptiert, wird dieser als exakte Übereinstimmung für den Variant-Typ angesehen.

Dies kann bei Gleitkommatypen einige geringe Nebeneffekte bewirken. Gleitkommatypen werden anhand der Größe verglichen. Falls es keine genaue Übereinstimmung für die Gleitkommavariablen gibt, die an den **overload**-Aufruf übergeben wird, aber ein varianter Parameter vorhanden ist, erhält die Variante den Vorzug gegenüber dem kleineren Gleitkommatyp.

Ein Beispiel:

```
procedure foo(i: integer); overload;
procedure foo(d: double); overload;
procedure foo(v: variant); overload;
var
  v: variant;
begin
  foo(1);      // Integer-Version
  foo(v);      // Variant-Version
  foo(1.2);    // Variant-Version (float literals -> extended precision)
end;
```

Dieses Beispiel ruft die **variant**-Version von **foo**, nicht die **double**-Version auf, da die Konstante 1.2 implizit ein **extended**-Typ ist. **extended** ist jedoch keine genaue Übereinstimmung für **double**. **extended** ist auch keine Übereinstimmung für **variant**, aber **variant** wird als allgemeinerer Typ angesehen (während **double** ein kleinerer Typ als **extended** ist).

```
foo(Double(1.2));
```

Diese Typumwandlung funktioniert nicht. Verwenden Sie stattdessen typisierte Konstanten.

```
const d: double = 1.2;
begin
  foo(d);
end;
```

Dieser Quelltext funktioniert korrekt und ruft die **double**-Version auf.

```
const s: single = 1.2;
begin
  foo(s);
end;
```

Dieser Quelltext funktioniert und ruft die **double**-Version von **foo** auf. **single** passt besser zu **double** als **variant**.

Bei der Deklaration von überladenen Routinen besteht die beste Möglichkeit zur Vermeidung einer Gleitkommepromotion zu Variant darin, eine Version ihrer überladenen Funktion für jeden Gleitkommatyp (**single**, **double**, **extended**) zusammen mit der

variant-Version zu deklarieren.

Bei Verwendung von Standardparametern in überladenen Routinen müssen Sie mehrdeutige Parametersignaturen vermeiden.

Die möglichen Auswirkungen des Überladens lassen sich beschränken, indem Sie beim Aufruf den qualifizierten Routinennamen verwenden. Mit `Unit1.MyProcedure(X, Y)` können nur die in `Unit1` deklarierten Routinen aufgerufen werden. Wenn der Name und die Parameterliste keiner Routine in `Unit1` entsprechen, gibt der Compiler einen Fehler aus.

Lokale Deklarationen

Der Rumpf einer Funktion oder Prozedur beginnt in vielen Fällen mit der Deklaration lokaler Variablen, die im Anweisungsblock der Routine verwendet werden. Sie können beispielsweise Konstanten, Typen und andere Routinen deklarieren. Der Gültigkeitsbereich lokaler Bezeichner ist auf die Routine beschränkt, in der sich die Deklaration befindet.

4

Verschachtelte Routinen

Funktionen und Prozeduren können im Abschnitt mit den lokalen Deklarationen ihrerseits Funktionen und Prozeduren enthalten. Die folgende Deklaration der Prozedur `DoSomething` enthält beispielsweise eine verschachtelte Prozedur:

```
procedure DoSomething(S: string);
var
  X, Y: Integer;

procedure NestedProc(S: string);
begin
  ...
end;

begin
  ...
  NestedProc(S);
  ...
end;
```

Der Gültigkeitsbereich einer verschachtelten Routine ist auf die Funktion bzw. Prozedur beschränkt, in der sie deklariert ist. Im letzten Beispiel kann `NestedProc` nur in `DoSomething` aufgerufen werden.

Echte Beispiele verschachtelter Routinen sind die Prozedur `DateTimeToString`, die Funktion `ScanDate` und andere Routinen in der Unit `SysUtils`.

Siehe auch

Parameter (siehe Seite 938)

Prozeduren und Funktionen aufrufen (siehe Seite 945)

4.1.1.5.2 Parameter

Dieses Thema enthält Informationen zu folgenden Bereichen:

- Parametersemantik
- String-Parameter
- Array-Parameter
- Standardparameter

Allgemeines zu Parametern

Die meisten Prozedur- und Funktions-Header enthalten eine Parameterliste. Im Header

```
function Power(X: Real; Y: Integer): Real;
```

lautet die Parameterliste beispielsweise `(X: Real; Y: Integer)`.

Eine Parameterliste ist eine Folge von Parameterdeklarationen, die durch Strichpunkte voneinander getrennt und in Klammern eingeschlossen werden. Jede Deklaration besteht aus einer Reihe von Parameternamen, die durch Kommas voneinander getrennt werden. Hinter den Parameternamen steht in den meisten Fällen ein Doppelpunkt und ein Typbezeichner, in Einzelfällen außerdem das Gleichheitszeichen (=) und ein Standardwert. Parameternamen müssen gültige Bezeichner sein. Jeder Deklaration kann **var**, **const** oder **out** vorausgehen. Beispiele:

```
(X, Y: Real)
(var S: string; X: Integer)
(HWnd: Integer; Text, Caption: PChar; Flags: Integer)
(const P; I: Integer)
```

Die Parameterliste gibt Anzahl, Reihenfolge und Typ der Parameter an, die beim Aufruf an die Routine übergeben werden müssen. Wenn eine Routine keine Parameter entgegennimmt, geben Sie in der Deklaration weder die Bezeichnerliste noch die Klammern an:

```
procedure UpdateRecords;
begin
  ...
end;
```

Im Rumpf der Prozedur oder Funktion können die Parameternamen (x und y im ersten Beispiel) als lokale Variablen eingesetzt werden. Sie dürfen die Parameternamen im Abschnitt mit den lokalen Deklarationen nicht erneut deklarieren.

Parametersemantik

Parameter können auf verschiedene Weise kategorisiert werden:

- Jeder Parameter ist als Wert-, Variablen-, Konstanten- oder Ausgabeparameter klassifiziert. Wertparameter werden verwendet, sofern nicht mit den reservierten Wörtern **var**, **const** und **out** einer der anderen Typen angegeben wird.
- Wertparameter sind immer typisiert, während Variablen-, Konstanten und Ausgabeparameter auch untypisiert sein können
- Für Array-Parameter gelten spezielle Regeln.

Dateien und Instanzen strukturierter Typen, die Dateien enthalten, können nur als Variablenparameter (**var**) übergeben werden.

Wert- und Variablenparameter

Die meisten Parameter sind Wert- (Standard) oder Variablenparameter (**var**). Wertparameter werden als Wert übergeben, Variablenparameter als Referenz. Der Unterschied wird anhand der folgenden beiden Funktionen deutlich:

```
function DoubleByValue(X: Integer): Integer; // X ist ein Wertparameter
begin
  X := X * 2;
  Result := X;
end;

function DoubleByRef(var X: Integer): Integer; // X ist ein Variablenparameter
begin
  X := X * 2;
  Result := X;
end;
```

Diese Funktionen liefern das gleiche Ergebnis. Nur die zweite Funktion (**DoubleByRef**) kann jedoch den Wert der an sie übergebenen Variablen ändern. Die Funktionen können beispielsweise folgendermaßen aufgerufen werden:

```
var
  I, J, V, W: Integer;
begin
  I := 4;
  V := 4;
  J := DoubleByValue(I); // J = 8, I = 4
  W := DoubleByRef(V); // W = 8, V = 8
end;
```

Nachdem diese Anweisungen ausgeführt wurden, enthält die an **DoubleByValue** übergebene Variable **I** immer noch den

ursprünglichen Wert. Die an `DoubleByRef` übergebene Variable `v` enthält dagegen einen neuen Wert.

Ein Wertparameter verhält sich wie eine lokale Variable, die durch den im Aufruf der Funktion oder Prozedur übergebenen Wert initialisiert wird. Wenn Sie eine Variable als Wertparameter übergeben, erstellt die Prozedur oder Funktion eine Kopie dieser Variablen. Änderungen des Wertes dieser Variablen wirken sich nicht auf die ursprüngliche Variable aus.

Sie werden verworfen, sobald die Steuerung wieder an den Aufrufer zurückgegeben wird. Ein Variablenparameter ist dagegen mit einem Zeiger vergleichbar. Änderungen des Parameters im Rumpf einer Prozedur oder Funktion bleiben deshalb erhalten, wenn die Programmausführung wieder dem Aufrufer übergeben wird und der Parametername nicht mehr im Gültigkeitsbereich liegt.

Auch wenn dieselbe Variable in mehreren `var`-Parametern übergeben wird, werden keine Kopien erstellt. Dieses Merkmal wird im folgenden Beispiel illustriert:

```
procedure AddOne(var X, Y: Integer);
begin
  X := X + 1;
  Y := Y + 1;
end;

var I: Integer;
begin
  I := 1;
  AddOne(I, I);
end;
```

Nachdem dieser Code ausgeführt wurde, enthält die Variable `I` den Wert 3.

Enthält die Deklaration einer Routine einen `var`-Parameter, müssen Sie im Aufruf der Routine einen Ausdruck übergeben, dem ein Wert zugewiesen werden kann, also eine Variable, eine typisierte Konstante (im Status `{$J+}`), einen dereferenzierten Zeiger, ein Feld oder eine indizierte Variable. Im Beispiel weiter oben würde `DoubleByRef(7)` einen Fehler generieren, während der Aufruf `DoubleByValue(7)` zulässig wäre.

In `var`-Parametern (beispielsweise `DoubleByRef(MyArray[I])`) übergebene Indizes und dereferenzierte Zeiger werden vor der Ausführung der Routine einmal ausgewertet.

Konstantenparameter

Ein Konstantenparameter (`const`) entspricht einer lokalen bzw. schreibgeschützten Variablen. Konstantenparameter entsprechen weitgehend Wertparametern. Sie können ihnen jedoch im Rumpf einer Prozedur oder Funktion keinen Wert zuweisen und sie nicht als `var`-Parameter an eine andere Routine übergeben. Übergeben Sie eine Objektreferenz als Konstantenparameter, können Sie weiterhin auf die Objekteigenschaften zugreifen und diese ändern.

Die Verwendung von `const` ermöglicht dem Compiler die Optimierung des Codes für strukturierte und String-Parameter. Gleichzeitig wird die versehentliche Übergabe eines Parameters als Referenz an eine andere Routine verhindert.

Das folgende Beispiel ist der Header der Funktion `CompareStr` in der Unit `SysUtils`:

```
function CompareStr(const S1, S2: string): Integer;
```

Da `S1` und `S2` im Rumpf von `CompareStr` nicht geändert werden, können sie als Konstantenparameter deklariert werden.

Ausgabeparameter

Ausgabeparameter werden mit dem Schlüsselwort `out` deklariert und wie Variablenparameter per Referenz übergeben. Der ursprüngliche Wert der referenzierten Variablen wird jedoch verworfen, wenn diese als Ausgabeparameter an eine Routine übergeben wird. Der Ausgabeparameter dient nur der Ausgabe. Er weist die Funktion oder Prozedur an, wo die Ausgabe zu speichern ist, stellt aber keinerlei Eingaben bereit.

Ein Beispiel ist der folgende Prozedur-Header:

```
procedure GetInfo(out Info: SomeRecordType);
```

Wenn Sie `GetInfo` aufrufen, müssen Sie eine Variable des Typs `SomeRecordType` übergeben.

```
var MyRecord: SomeRecordType;
  ...
GetInfo(MyRecord);
```

In `MyRecord` werden jedoch keine Daten an die Prozedur `GetInfo` übergeben. `MyRecord` dient nur als Container für die von `GetInfo` generierten Informationen. Beim Aufruf von `GetInfo` wird der von `MyRecord` belegte Speicher sofort freigegeben, noch bevor die Steuerung an die Prozedur übergeben wird.

Ausgabeparameter werden häufig in verteilten Objektmodellen wie COM eingesetzt. Sie sollten diesen Parametertyp auch verwenden, wenn Sie nicht initialisierte Variablen an Funktionen oder Prozeduren übergeben.

Untypisierte Parameter

Variablen-, Konstanten- und Ausgabeparameter (`var`, `const` und `out`) müssen im Gegensatz zu Wertparametern nicht typisiert werden. Zum Beispiel:

```
procedure TakeAnything(const C);
```

Diese Anweisung deklariert eine Prozedur namens `TakeAnything`, die einen Parameter beliebigen Typs entgegennimmt. Im Rumpf einer Prozedur oder Funktion sind untypisierte Parameter mit keinem Typ kompatibel.

Bevor der untypisierte Parameter bearbeitet werden kann, muss eine Typkonvertierung erfolgen. Der Compiler kann nicht feststellen, ob Operationen, die mit untypisierten Parametern durchgeführt werden, zulässig sind.

Die folgende Beispielfunktion `Equal` vergleicht die angegebene Anzahl Byte in zwei beliebigen Variablen, die als untypisierte Parameter übergeben werden:

```
function Equal(var Source, Dest; Size: Integer): Boolean;
type
  TBytes = array[0..MaxInt - 1] of Byte;
var
  N : Integer;
begin
  N := 0;
  while (N < Size) and (TBytes(Dest)[N] = TBytes(Source)[N]) do
    Inc(N);
  Equal := N = Size;
end;
```

Nach den Deklarationen

```
type
  TVector = array[1..10] of Integer;
  TPoint = record
    X, Y: Integer;
  end;
var
  Vec1, Vec2: TVector;
  N : Integer;
  P: TPoint;
```

kann die Funktion `Equal` folgendermaßen aufgerufen werden:

```
Equal(Vec1, Vec2, SizeOf(TVector));      // Vec1 wird mit Vec2 verglichen
Equal(Vec1, Vec2, SizeOf(Integer) * N); // Die ersten N Elemente von Vec1 und Vec2 vergleichen
Equal(Vec1[1], Vec1[6], SizeOf(Integer) * 5); // Die ersten fünf mit den letzten fünf
Elementen von Vec1 vergleichen
Equal(Vec1[1], P, 4);                  // Vec1[1] mit P.X und Vec1[2] mit P.Y vergleichen
```

String-Parameter

Bei der Deklaration von Routinen, die kurze Strings als Parameter entgegennehmen, können Sie in den Parameterdeklarationen keine Längenangaben verwenden. Ein Beispiel:

```
procedure Check(S: string[20]); // Syntaxfehler
```

Diese Deklaration führt zu einem Compilierungsfehler. Dagegen wird beim Ausdruck

```
type TString20 = string[20];
procedure Check(S: TString20);
```

ist zulässig. Mit dem speziellen Bezeichner **OpenString** können Sie Routinen deklarieren, die kurze Strings unterschiedlicher Länge als Parameter entgegennehmen:

```
procedure Check(S: OpenString);
```

Wenn die Compiler-Direktiven `{$H}` und `{$P+}` in Kraft sind, ist das reservierte Wort **string** in Parameterdeklarationen gleichbedeutend mit **OpenString**.

Kurze Strings, **OpenString**, `$_H` und `$_P` werden nur aus Gründen der Abwärtskompatibilität unterstützt. In neuem Quelltext sollten Sie immer lange Strings verwenden.

4 Array-Parameter

Bei der Deklaration von Routinen, die Array-Parameter entgegennehmen, können Sie in den Parameterdeklarationen keinen Indextyp angeben. Ein Beispiel:

```
procedure Sort(A: array[1..10] of Integer) // Syntaxfehler
```

Diese Deklaration führt zu einem Compilierungsfehler. Dagegen wird beim Ausdruck

```
type TDigits = array[1..10] of Integer;
procedure Sort(A: TDigits);
```

ist zulässig. Eine andere Möglichkeit ist die Verwendung von offenen Array-Parametern.

Da in Delphi keine Wertesemantik für dynamische Arrays implementiert ist, stellen "Wert"-Parameter in Routinen keine vollständige Kopie des dynamischen Arrays dar. Beispiel:

```
type
  TDynamicArray = array of Integer;
  procedure p(Value: TDynamicArray);
begin
  Value[0] := 1;
end;

  procedure Run;
var
  a: TDynamicArray;
begin
  SetLength(a, 1);
  a[0] := 0;
  p(a);
  Writeln(a[0]); // Gibt '1' aus
end;
```

Beachten Sie, dass die Zuweisung zu `Value[0]` in Routine `p` den Inhalt des dynamischen Arrays der aufrufenden Routine ändert, obwohl `Value` ein Wertparameter ist. Falls eine vollständige Kopie des dynamischen Arrays erforderlich ist, verwenden Sie die Standardprozedur `Copy`, um eine Wertekopie des dynamischen Arrays zu erstellen.

Offene Array-Parameter

Offene Array-Parameter ermöglichen die Übergabe von Arrays unterschiedlicher Größe an dieselbe Funktion oder Prozedur. Die Definition eines offenen Array-Parameters erfolgt mit der Syntax `array of Typ` (anstelle der Syntax `array[X..Y] of Typ`) in der Parameterdeklaration. Zum Beispiel:

```
function Find(A: array of Char): Integer;
```

Hier wird eine Funktion namens `Find` deklariert, die ein Zeichen-Array beliebiger Größe entgegennimmt und einen Integer-Wert zurückgibt.

Anmerkung: Die Syntax offener Array-Parameter erinnert an dynamische Arrays, obwohl diese beiden Array-Typen nicht identisch sind. Das obige Beispiel deklariert eine Funktion, die ein beliebiges Zeichen-Array entgegennimmt, also auch ein dynamisches Array. Die Deklaration eines Parameters, bei dem es sich um ein dynamisches Array handeln soll, muss mit Angabe eines Typbezeichners erfolgen:

```
type TDYNAMICCharArray = array of Char;
function Find(A: TDYNAMICCharArray): Integer;
```

Im Rumpf einer Routine gelten die folgenden Regeln für offene Array-Parameter.

- Sie sind stets nullbasiert. Das erste Element trägt immer die Indexnummer 0, das zweite die 1 usw. Die Standardfunktionen **Low** und **High** geben 0 bzw. *Length* - 1 zurück. Die Funktion **SizeOf** gibt die Größe des Arrays zurück, das an die Routine übergeben wird.
- Der Zugriff kann nur auf die einzelnen Elemente erfolgen. Zuweisungen an einen offenen Array-Parameter insgesamt sind dagegen nicht zulässig.
- Sie können nur als offene Array-Parameter oder als untypisierte Variablenparameter (**var**) an andere Prozeduren und Funktionen übergeben werden. Eine Übergabe an *SetLength* ist nicht möglich.
- Anstelle eines Arrays können Sie eine Variable mit dem Basistyp des offenen Array-Parameters übergeben, die dann wie ein Array der Länge 1 verarbeitet wird.

Wenn Sie ein Array als offenen Array-Wertparameter übergeben, erstellt der Compiler im Stack-Bereich der Routine eine lokale Kopie des Arrays. Die Übergabe großer Parameter kann also zu einem Stack-Überlauf führen.

Die folgenden Beispiele verwenden offene Array-Parameter, um eine Prozedur namens **Clear** zu definieren, die jedem Element eines Arrays mit reellen Zahlen den Wert Null zuweist. Außerdem wird die Funktion **Sum** definiert, mit der die Summe der Elemente in einem Array mit reellen Zahlen ermittelt werden kann.

```
procedure Clear(var A: array of Real);
var
  I: Integer;
begin
  for I := 0 to High(A) do A[I] := 0;
end;

function Sum(const A: array of Real): Real;
var
  I: Integer;
  S: Real;
begin
  S := 0;
  for I := 0 to High(A) do S := S + A[I];
  Sum := S;
end;
```

Wenn Sie Routinen aufrufen, die offene Array-Parameter verarbeiten, können Sie offene Array-Konstruktoren übergeben.

Variante offene Array-Parameter

Variante offene Array-Parameter ermöglicht die Übergabe eines Arrays mit Ausdrücken unterschiedlicher Typen an eine Prozedur oder Funktion. In der Definition einer Routine mit einem variablen offenen Array-Parameter geben Sie als Typ des Parameters **array of const** an. Die Anweisung:

```
procedure DoSomething(A: array of const);
```

deklariert eine Prozedur namens **DoSomething**, die auch heterogene Arrays verarbeiten kann.

Auf der .NET-Plattform entspricht ein variabler offener Array-Parameter **array of TObject**. Um den Typ eines Array-Elements zu bestimmen, können Sie die Methoden **ClassName** oder **GetType** verwenden.

Auf der Win32-Plattform ist die Konstruktion **array of const** zur Konstruktion **array of TVarRec** äquivalent. **TVarRec** ist in der Unit **System** deklariert und repräsentiert einen Record mit variablen Bestandteilen, der Werte der Typen **Integer**, **Boolean**, **Zeichen**, **Real**, **String**, **Zeiger**, **Klasse**, **Klassenreferenz**, **Interface** und **Variant** aufnehmen kann. Das Feld **vType** im Record

TVarRec gibt den Typ der einzelnen Elemente im Array an. Einige Typen werden nicht per Wert, sondern per Referenz übergeben. Insbesondere lange Strings werden als **Pointer** übergeben und müssen in den Typ **String** umgewandelt werden.

Das folgende Win32-Beispiel verwendet einen varianten offenen Array-Parameter in einer Funktion, die aus jedem Element im Array einen String erzeugt. Die einzelnen Strings werden dann verkettet. Die in dieser Funktion aufgerufenen Routinen zur String-Verarbeitung sind in SysUtils definiert. Diese Funktion wird unter .NET nicht kompiliert, weil sie von der varianten Implementierung von `array of const` abhängt.

```
function MakeStr(const Args: array of const): string;
var
  I: Integer;
begin
  Result := '';
  for I := 0 to High(Args) do
    with Args[I] do
      case VType of
        vtInteger: Result := Result + IntToStr(VInteger);
        vtBoolean: Result := Result + BoolToStr(VBoolean);
        vtChar: Result := Result + VChar;
        vtExtended: Result := Result + FloatToStr(VExtended^);
        vtString: Result := Result + VString^;
        vtPChar: Result := Result + VPChar;
        vtObject: Result := Result + VObject.ClassName;
        vtClass: Result := Result + VClass.ClassName;
        vtAnsiString: Result := Result + string(VAnsiString);
        vtCurrency: Result := Result + CurrToStr(VCurrency^);
        vtVariant: Result := Result + string(VVariant^);
        vtInt64: Result := Result + IntToStr(VInt64^);
      end;
  end;
end;
```

Der Aufruf dieser Funktion kann mit einem offenen Array-Konstruktor erfolgen. Beispiel:

```
MakeStr(['test', 100, ' ', True, 3.14159, TForm])
```

Dieser Aufruf gibt den String 'test100 T3.14159TForm' zurück.

Standardparameter

Sie können im Header einer Prozedur oder Funktion Standardparameterwerte angeben. Standardwerte sind nur für typisierte Konstanten- und für Wertparameter zulässig. Die Angabe des Standardwertes erfolgt mit dem Gleichheitszeichen (=) hinter der Parameterdeklaration und einem Konstantenausdruck, der zum Typ des Parameters zuweisungskompatibel ist.

Betrachten Sie beispielsweise die folgende Deklaration:

```
procedure FillArray(A: array of Integer; Value: Integer = 0);
```

Nach dieser Deklaration sind die folgenden Prozeduraufrufe äquivalent:

```
FillArray(MyArray);
  FillArray(MyArray, 0);
```

In einer Deklaration mehrerer Parameter kann kein Standardwert angegeben werden. Die Deklaration

```
function MyFunction(X: Real = 3.5; Y: Real = 3,5): Real;
```

ist zulässig, während

```
function MyFunction(X, Y: Real = 3,5): Real; // Syntaxfehler
```

nicht korrekt ist.

Parameter mit Standardwerten müssen am Ende der Parameterliste angegeben werden. Sobald einem Parameter ein Standardwert zugewiesen wurde, müssen Sie auch allen folgenden Parametern Standardwerte zuweisen. Die folgende Deklaration ist aus diesem Grund nicht zulässig:

```
procedure MyProcedure(I: Integer = 1; S: string); // Syntaxfehler
```

In einem prozeduralen Typ angegebene Werte überladen die in einer Routine angegeben Werte. In den Deklarationen

```
type TResizer = function(X: Real; Y: Real = 1,0): Real;
function Resizer(X: Real; Y: Real = 2.0): Real;
var
  F: TResizer;
  N : Real;
```

führen die Anweisungen

```
F := Resizer;
F(N);
```

zur Übergabe der Werte (N, 1.0) an Resizer.

Für Standardparameter dürfen nur Werte verwendet werden, die in Form eines Konstantenausdrucks angegeben werden können. Für prozedurale Parameter oder Parameter vom Typ dynamisches Array, Klasse, Klassenreferenz oder Interface kann deshalb nur der Standardwert **nil** verwendet werden. Für Parameter vom Typ Record, Variant, Datei, statisches Array oder Objekt sind keine Standardwerte zulässig.

Standardparameter und überladene Funktionen

Wenn Sie Standardparameterwerte in überladenen Routinen einsetzen, müssen Sie mehrdeutige Parametersignaturen vermeiden. Ein Beispiel:

```
procedure Confused(I: Integer); overload;
  ...
procedure Confused(I: Integer; J: Integer = 0); overload;
  ...
Confused(X); // Welche Prozedur wird aufgerufen?
```

Tatsächlich wird keine der beiden Prozeduren aufgerufen. Diese Zeilen führen zu einem Compilierungsfehler.

Standardparameter in forward- und interface-Deklarationen

Wenn eine Routine eine **forward**-Deklaration enthält oder im **interface**-Abschnitt einer Unit definiert ist, können Sie die Standardparameterwerte nur in der **forward**- bzw. **interface**-Deklaration angeben. Standardwerte in der definierenden Deklaration werden ignoriert. Liegt für eine Routine keine **forward**- oder **interface**-Deklaration vor, kann die definierende Deklaration Standardparameterwerte angeben.

Siehe auch

Prozeduren und Funktionen (siehe Seite 931)

Prozeduren und Funktionen aufrufen (siehe Seite 945)

4.1.1.5.3 Prozeduren und Funktionen aufrufen

Dieses Thema enthält Informationen zu folgenden Bereichen:

- Ablaufsteuerung und Parameter in Routinen
- Offene Array-Konstruktoren
- Die **inline**-Direktive

Auflaufsteuerung und Parameter

Wenn Sie eine Prozedur oder Funktion aufrufen, wird die Steuerung vom Punkt des Aufrufs an den Rumpf der Routine übergeben. Der Aufruf kann mit dem deklarierten Namen der Routine (mit oder ohne Qualifizierer) oder mit einer prozeduralen Variablen erfolgen, die auf die Routine zeigt. In beiden Fällen müssen im Aufruf Parameter übergeben werden, die in der Reihenfolge und im Typ den in der Parameterliste der Routine angegeben Parametern entsprechen. Die an eine Routine übergebenen Parameter werden auch als tatsächliche Parameter bezeichnet - im Gegensatz zu den formalen Parametern in der

Deklaration der Routine.

Beachten Sie beim Aufrufen einer Routine Folgendes:

- Ausdrücke zur Übergabe typisierter **const**- und Wertparameter müssen zu den entsprechenden formalen Parametern zuweisungskompatibel sein.
- Ausdrücke zur Übergabe von **var**- und **out**-Parametern müssen gegebenenfalls wie die entsprechenden formalen Parameter typisiert sein.
- **var**- und **out**-Parameter können nur in zuweisungsfähigen Ausdrücken übergeben werden.
- Wenn die formalen Parameter einer Routine nicht typisiert sind, dürfen Zahlen und echte Konstanten mit numerischen Werten nicht als tatsächliche Parameter verwendet werden.

Wenn Sie eine Routine mit Standardparameterwerten aufrufen, müssen für alle Parameter nach dem ersten akzeptierten Standardwert ebenfalls Standardwerte existieren. Aufrufe der Form `SomeFunction(, , X)` sind nicht zulässig.

Nimmt eine Routine Parameter entgegen, müssen Sie im Aufruf die Klammern angeben, auch wenn alle tatsächlichen Parameter Standardwerte sind. Um beim Aufruf der Prozedur

```
procedure DoSomething(X: Real = 1.0; I: Integer = 0; S: string = ''');
```

alle Standardwerte zu übernehmen, geben Sie Folgendes ein:

```
DoSomething();
DoSomething;
```

Offene Array-Konstruktoren

Offene Array-Konstruktoren ermöglichen die Bildung von Arrays im Aufruf einer Funktion oder Prozedur. Sie müssen als offene Array-Parameter oder variante offene Array-Parameter übergeben werden.

Ein offener Array-Konstruktor ist wie ein Mengenkonstruktor eine Folge von Ausdrücken, die durch Kommas voneinander getrennt und in eckigen Klammern angegeben werden.

Ausgehend von den Deklarationen

```
var I, J: Integer;
    procedure Add(A: array of Integer);
```

können Sie die Prozedur `Add` mit folgender Anweisung aufrufen:

```
Add([5, 7, I, I + J]);
```

Dieses Verfahren ist zum Folgenden äquivalent:

```
var Temp: array[0..3] of Integer;
    ...
    Temp[0] := 5;
    Temp[1] := 7;
    Temp[2] := I;
    Temp[3] := I + J;
    Add(Temp);
```

Offene Array-Konstruktoren können nur als Wert- oder **const**-Parameter übergeben werden. Die Ausdrücke im Konstruktor müssen zum Basistyp des Array-Parameters zuweisungskompatibel sein. Wird ein varianter offener Array-Parameter verwendet, können die Ausdrücke unterschiedliche Typen aufweisen.

Die Direktive **Inline**

Beim Delphi-Compiler ermöglicht zur Verbesserung der Leistung, Funktionen und Prozeduren mit der Direktive **inline** zu versehen. Wenn eine Funktion oder Prozedur bestimmten Kriterien entspricht, fügt der Compiler Code direkt ein anstatt einen Aufruf zu generieren. Das Ergebnis dieser Leistungsoptimierung ist schnellerer Code, der jedoch mehr Speicherplatz in Anspruch nimmt. Der Compiler produziert dabei eine größere Binärdatei. Die Direktive **inline** wird in Funktions- und Prozedurdeklarationen und -definitionen, genau wie andere Direktiven, verwendet.

```

procedure MyProc(x:Integer); inline;
begin
  // ...
end;

function MyFunc(y:Char) : String; inline;
begin
  // ...
end;

```

Die Direktive **inline** muss als Vorschlag an den Compiler gesehen werden. Es gibt keine Garantie, dass der Compiler eine Routine als Inline übernimmt, da bestimmte Umstände dies verhindern können. Die folgende Liste enthält die Bedingungen, unter welchen ein Inlining vorgenommen bzw. nicht vorgenommen wird.

- Inlining wird für keinerlei spät-gebundene Methoden nicht vorgenommen. Dazu gehören virtuelle, dynamische und Botschaftsmethoden.
- Für Routinen, die Assembler-Code enthalten, wird kein Inlining vorgenommen.
- Für Konstruktoren und Destruktoren wird kein Inlining vorgenommen.
- Für den Hauptprogrammblock, die Unit-Initialisierungs- und -Finalisierungsblöcke wird kein Inlining vorgenommen.
- Für Routinen, die vor der Verwendung nicht definiert wurden, kann kein Inlining vorgenommen werden.
- Für Routinen, die offene Array-Parameter übernehmen, wird kein Inlining vorgenommen.
- Für Code in Packages kann ein Inlining vorgenommen werden, Inlining über Packages-Grenzen ist jedoch nicht möglich.
- Für Units, die zirkulär abhängig sind, wird kein Inlining vorgenommen. Dazu gehören auch indirekte, zirkuläre Abhängigkeiten, wie z.B. Unit A verwendet Unit B und Unit B verwendet Unit C, die wiederum Unit A verwendet. In diesem Beispiel wird beim Compilieren von Unit A kein Inlining von Unit B oder Unit C in Unit A vorgenommen.
- Der Compiler kann für Code ein Inlining vornehmen, wenn eine Unit zirkulär abhängig ist, sofern der betreffende Code aus einer Unit außerhalb der zirkulären Beziehung stammt. Wenn im obigen Beispiel Unit A auch Unit D verwendet, könnte für Code aus Unit D ein Inlining vorgenommen werden, da Unit D nicht in die zirkuläre Abhängigkeit einbezogen ist.
- Wenn eine Routine im **interface**-Abschnitt definiert ist, die auf im **implementation**-Abschnitt definierte Symbole zugreift, dann kann für diese Routine kein Inlining vorgenommen werden.
- In Delphi.NET kann für Routinen in Klassen kein Inlining vorgenommen werden, wenn sie auf Elemente mit geringer (eingeschränkterer) Sichtbarkeit als die Methode selbst zugreifen. Wenn beispielsweise eine als **public** deklarierte Methode auf als **private** deklarierte Symbole zugreift, kann für die Methode kein Inlining vorgenommen werden.
- Wenn eine mit der Direktive **inline** gekennzeichnete Routine externe Symbole aus anderen Units verwendet, müssen diese Units alle in der **uses**-Anweisung enthalten sein, ansonsten kann kein Inlining vorgenommen werden.
- Für Prozeduren und Funktionen, die in konditionalen Ausdrücken in **while-do-** und **repeat-until**-Anweisungen verwendet werden, kann kein Inlining vorgenommen werden.
- In einer Unit sollte der Rumpf für eine Inline-Funktion definiert sein, bevor die Funktion aufgerufen wird. Ansonsten kann der Rumpf der Funktion, der dem Compiler bei Erreichen der Aufrufposition nicht bekannt ist, nicht als inline expandiert werden.

Wenn Sie die Implementierung von mit der Direktive **inline** versehenen Routinen ändern, werden alle Units, die diese Funktionen verwenden, neu compiliert. Dies ist ein Unterschied zu den herkömmlichen Regeln für das Neucompilieren, da eine Neucomplierung früher nur durch Änderungen im **interface**-Abschnitt einer Unit ausgelöst wurde.

Durch die **{\$INLINE}**-Compiler-Direktive wird Ihnen eine besserer Steuerung des Inlining ermöglicht. Die Direktive **{\$INLINE}** kann sowohl bei der Definition einer Inline-Routine als auch bei deren Aufruf eingesetzt werden. Im Folgenden finden Sie die möglichen Werte und deren Bedeutung:

Wert	Bedeutung bei der Definition	Bedeutung beim Aufruf
{\$INLINE ON} (Vorgabe)	Die Routine wird als Kandidat für das Inlining compiliert, wenn sie mit der inline -Direktive versehen ist.	Falls möglich, wird für die Routine wird ein Inlining vorgenommen.

{\$INLINE AUTO}	Verhält sich wie {\$INLINE ON}; zusätzlich werden Routinen ohne die Direktive inline für das Inlining vorgesehen, wenn ihr Code weniger oder genau 32 Bytes umfasst.	{\$INLINE AUTO} hat keine Auswirkungen darauf, ob für eine Routine Inlining angewendet wird, wenn es auf der Aufrufseite der Routine verwendet wird.
{\$INLINE OFF}	Die Routine wird nicht als Kandidat für das Inlining vorgesehen, selbst wenn sie mit der Direktive inline gekennzeichnet ist.	Für die Routine wird kein Inlining vorgenommen.

Siehe auch

Prozeduren und Funktionen (siehe Seite 931)

Parameter (siehe Seite 938)

4.1.1.6 Klassen und Objekte

In diesem Abschnitt erfahren Sie Näheres zu den objektorientierten Leistungsmerkmalen der Sprache Delphi. Hierzu gehört beispielsweise die Deklaration und Verwendung von Klassentypen.

4.1.1.6.1 Klassen und Objekte

In diesem Thema werden folgende Bereiche erläutert:

- Deklarationssyntax von Klassen
- Vererbung und Gültigkeitsbereich
- Sichtbarkeit von Klassenelementen
- Vorwärtsdeklarationen und voneinander abhängige Klassen

Klassentypen

Eine Klasse (oder ein Klassentyp) definiert eine Struktur von Feldern, Methoden und Eigenschaften. Die Instanzen eines Klassentyps heißen Objekte. Die Felder, Methoden und Eigenschaften einer Klasse nennt man ihre Komponenten oder Elemente.

- Felder sind im Wesentlichen Variablen, die zu einem Objekt gehören. Sie definieren Datenelemente, die in jeder Instanz der Klasse vorhanden sind.
- Eine Methode ist eine Prozedur oder Funktion, die zu einer bestimmten Klasse gehört. Die meisten Methoden führen Operationen mit Objekten (Instanzen einer Klasse) durch. Manche Methoden arbeiten jedoch mit den Klassentypen selbst. Sie werden als Klassenmethoden bezeichnet.
- Eine Eigenschaft ist ein Interface zu den Daten eines Objekts (die oftmals in einem Feld gespeichert sind). Eigenschaften verfügen über Zugriffsbezeichner, die bestimmen, wie ihre Daten gelesen und geändert werden. Sie erscheinen für die anderen Bestandteile eines Programms (außerhalb des Objekts) in vielerlei Hinsicht wie ein Feld.

Objekte sind dynamisch zugewiesene Speicherblöcke, deren Struktur durch ihren Klassentyp festgelegt wird. Jedes Objekt verfügt über eine eigene Kopie der in der Klasse definierten Felder. Die Methoden werden jedoch von allen Instanzen gemeinsam genutzt. Das Erstellen und Freigeben von Objekten erfolgt mithilfe spezieller Methoden, den Konstruktoren und Destruktoren.

Eine Klassentypvariable ist eigentlich ein Zeiger auf ein Objekt. Aus diesem Grund können auch mehrere Variablen auf dasselbe Objekt verweisen. Klassentypvariablen können wie andere Zeiger den Wert **nil** annehmen. Sie müssen aber nicht explizit dereferenziert werden, um auf das betreffende Objekt zuzugreifen. So wird beispielsweise durch die Anweisung `MeinObjekt.Size := 100` der Eigenschaft `Size` des Objekts, auf das `MeinObjekt` zeigt, der Wert 100 zugewiesen. Sie brauchen in diesem Fall nicht `MeinObjekt^.Size := 100` anzugeben.

Ein Klassentyp muss deklariert und benannt werden, damit er instantiiert werden kann (er kann also nicht in einer Variablen Deklaration definiert werden). Deklarieren Sie Klassen nur im äußersten Gültigkeitsbereich eines Programms oder einer Unit, nicht in einer Prozedur oder Funktion.

Ein Klassentyp wird folgendermaßen deklariert:

```
type
  Klassenname = class [abstract | sealed] (Vorfahrklasse)
    Elementliste
  end;
```

Klassenname ist ein beliebiger, gültiger Bezeichner, die Schlüsselwörter **sealed** und **abstract** sind optional, (*Vorfahrklasse*) ist optional, und *Elementliste* definiert die Felder, Methoden und Eigenschaften der Klasse. Wird keine *Vorfahrklasse* angegeben, erbt die neue Klasse direkt von der vordefinierten Basisklasse `TObject`. Wenn Sie eine *Vorfahrklasse* angeben und die *Elementliste* leer ist, brauchen Sie **end** nicht anzugeben. Eine Klassentypdeklaration kann auch eine Liste von Interfaces enthalten, die von der Klasse implementiert werden (siehe Interfaces implementieren (siehe Seite 1009)).

Wenn eine Klasse als **sealed** gekennzeichnet ist, dann kann sie nicht durch Vererbung erweitert werden. Ist eine Klasse als **abstract** markiert, dann kann sie nicht direkt mit dem Konstruktor `Create` instantiiert werden. Eine gesamte Klasse kann als **abstract** deklariert werden, auch wenn sie keine abstrakten virtuellen Methoden (siehe Seite 956) enthält. Eine Klasse kann nicht gleichzeitig **abstract** und **sealed** sein.

Methoden werden in einer Klassendeklaration als Funktions- oder Prozedurköpfe ohne Rumpf angegeben. Die definierenden Deklarationen folgen dann an einer anderen Stelle im Programm.

Das folgende Beispiel zeigt die Deklaration der Klasse `TMemoryStream` in der Unit `Classes`.

```
type TMemoryStream = class(TCustomMemoryStream)
  private
    FCapacity: Longint;
    procedure SetCapacity(NewCapacity: Longint);
  protected
    function Realloc(var NewCapacity: Longint): Pointer; virtual;
    property Capacity: Longint read FCapacity write SetCapacity;
  public
    destructor Destroy; override;
    procedure Clear;
    procedure LoadFromStream(Stream: TStream);
    procedure LoadFromFile(const FileName: string);
    procedure SetSize(NewSize: Longint); override;
    function Write(const Buffer; Count: Longint): Longint; override;
  end;
```

`TMemoryStream` ist von `TCustomMemoryStream` (in der Unit `Classes`) abgeleitet und erbt die meisten Elemente dieser Klasse. Zusätzlich werden jedoch mehrere Methoden und Eigenschaften einschließlich des Destruktors `Destroy` definiert bzw. neu definiert. Der Konstruktor `Create` wird ohne Änderung von `TObject` übernommen und daher nicht neu deklariert. Die verschiedenen Elemente sind als **private**, **protected** oder **public** deklariert (diese Klasse verfügt über keine **published**-Elemente). Informationen zu diesen Sichtbarkeitsangaben finden Sie weiter unten.

Ausgehend von dieser Deklaration kann eine Instanz von `TMemoryStream` folgendermaßen erstellt werden:

```
var stream: TMemoryStream;
  stream := TMemoryStream.Create;
```

Vererbung und Gültigkeitsbereich

Beim Deklarieren einer Klasse kann der direkte Vorfahr angegeben werden. Beispiel:

```
type TSomeControl = class(TControl);
```

Hier wird die Klasse `TSomeControl` von `TControl` abgeleitet. Ein Klassentyp erbt automatisch alle Elemente seines direkten Vorfahren. Außerdem können jederzeit neue Elemente erstellt oder geerbte Elemente neu definiert werden. Es ist aber nicht möglich, Elemente zu entfernen, die in einer Vorfahrklasse definiert wurden. Aus diesem Grund enthält `TSomeControl` alle in

der Klasse `TControl` und deren Vorfahren definierten Elemente.

Der Gültigkeitsbereich eines Elementbezeichners reicht von seiner Deklaration bis zum Ende der Klassendeklaration und erstreckt sich über alle Nachkommen des Klassentyps und alle in der Klasse und ihren Nachfahren definierten Methoden.

TObject und TClass

Die in der Unit `System` deklarierte Klasse `TObject` ist der absolute Vorfahr aller anderen Klassentypen. In der Klasse `TObject` sind nur einige wenige Methoden definiert, zu denen ein einfacher Konstruktor und ein Destruktor gehören. Außer `TObject` ist in `System` auch noch der Klassenreferenztyp (siehe Seite 974) `TClass` deklariert:

```
4
TClass = class of TObject;
```

Wenn Sie in der Deklaration eines Klassentyps keinen Vorfahren angegeben, erbt die Klasse direkt von `TObject`. Die Anweisung:

```
type TMyClass = class
  ...
end;
```

ist gleichbedeutend mit

```
type TMyClass = class(TObject)
  ...
end;
```

Die zweite Variante verdient jedoch aus Gründen der Lesbarkeit den Vorzug.

Kompatibilitätsregeln für Klassentypen

Ein Klassentyp ist mit jedem seiner Vorfahren zuweisungskompatibel. Eine Klassentypvariable kann daher auf eine Instanz einer beliebigen übergeordneten Klasse verweisen. Ausgehend von den Deklarationen

```
type
  TFigure = class(TObject);
  TRectangle = class(TFigure);
  TSquare = class(TRectangle);
var
  Fig: TFigure;
```

können der Variablen `Fig` Instanzen von `TFigure`, `TRectangle` und `TSquare` zugewiesen werden.

Objekttypen

Der Win32-Compiler von Delphi unterstützt als Alternative zu Klassentypen die Deklaration von Objekttypen. Die Syntax hierfür lautet:

```
type Objekttypname = object (VorfahrObjekttyp)
  Elementliste
end;
```

`Objekttypname` ist ein beliebiger gültiger Bezeichner, (`VorfahrObjekttyp`) ist optional, und `Elementliste` definiert die Felder, Methoden und Eigenschaften der Klasse. Wird kein `VorfahrObjekttyp` angegeben, hat der neue Typ keinen Vorfahren. Bei Objekttypen können Elemente nicht als **published** deklariert werden.

Da Objekttypen nicht von `TObject` abgeleitet sind, verfügen sie über keine integrierten Konstruktoren, Destruktoren oder andere Methoden. Instanzen können mit der Prozedur `New` erstellt und mit `Dispose` freigegeben werden. Sie können Variablen eines Objekttyps aber auch einfach wie bei einem Record-Typ deklarieren.

Objekttypen werden nur aus Gründen der Abwärtskompatibilität unterstützt. Ihre Verwendung unter Win32 ist nicht zu empfehlen. Vom Delphi .NET-Compiler werden sie nicht akzeptiert.

Sichtbarkeit von Klassenelementen

In einer Klasse hat jedes Element ein Sichtbarkeitsattribut, das durch die reservierten Wörter **private**, **protected**, **public**,

published oder **automated** angegeben wird. Zum Beispiel:

```
published property Color: TColor read GetColor write SetColor;
```

In diesem Beispiel wird die Eigenschaft `Color` als **published** deklariert. Die Sichtbarkeit bestimmt, wo und wie auf ein Element zugegriffen werden kann. **private** entspricht der geringsten, **protected** einer mittleren und **public**, **published** und **automated** der größten Sichtbarkeit.

Ein Element ohne Attribut erhält automatisch die Sichtbarkeit des vorhergehenden Elements in der Deklaration. Die Elemente am Anfang einer Klassendeklaration ohne explizite Sichtbarkeitsangabe werden standardmäßig als **published** deklariert, wenn die Klasse im Status `{$M+}` kompiliert oder von einer mit `{$M+}` kompilierten Klasse abgeleitet wurde. Andernfalls erhalten sie das Attribut **public**.

Aus Gründen der Lesbarkeit sollten Sie die Elemente einer Klassendeklaration nach ihrer Sichtbarkeit gruppieren. Nehmen Sie daher zuerst alle **private**-Elemente, anschließend alle **protected**-Elemente usw. in die Deklaration auf. Bei dieser Vorgehensweise braucht das Sichtbarkeitsattribut nur einmal angegeben zu werden, und es markiert immer den Anfang eines neuen Deklarationsabschnitts. Eine typische Klassendeklaration sieht somit folgendermaßen aus:

```
type
  TMyClass = class(TControl)
  private
    ... { private-Deklarationen }
  protected
    ... { protected-Deklarationen }
  public
    ... { public-Deklarationen }
  published
    ... { published-Deklarationen }
end;
```

Sie können die Sichtbarkeit eines Elements in einer untergeordneten Klasse durch Neudeklarieren erhöhen, jedoch nicht verringern. So kann beispielsweise eine **protected**-Eigenschaft in einer abgeleiteten Klasse als **public** deklariert werden, nicht aber als **private**. Außerdem können **published**-Elemente nicht zu **public**-Elementen gemacht werden. Weitere Informationen hierzu finden Sie unter Eigenschaften überschreiben und neu deklarieren (siehe Seite 965).

private-, protected- und public-Elemente

Auf ein **private**-Element kann nur innerhalb des Moduls (Unit oder Programm) zugegriffen werden, in dem die Klasse deklariert ist. Mit anderen Worten: Eine **private**-Methode kann nicht von anderen Modulen aufgerufen werden, und als **private** deklarierte Felder oder Eigenschaften können nicht von anderen Modulen gelesen oder geschrieben werden. Indem Sie verwandte Klassendeklarationen im selben Modul zusammenfassen, können Sie diesen Klassen also den Zugriff auf alle **private**-Elemente ermöglichen, ohne die Elemente anderen Modulen bekannt zu machen.

Ein **protected**-Element ist innerhalb des Moduls mit der Klassendeklaration und in allen abgeleiteten Klassen (unabhängig davon, in welchem Modul sie deklariert sind) sichtbar. Auf ein **protected**-Element können alle Methoden einer Klasse zugreifen, die von der Klasse mit der Elementdeklaration abgeleitet ist. Mit diesem Sichtbarkeitsattribut werden also Elemente deklariert, die nur in den Implementierungen abgeleiteter Klassen verwendet werden sollen.

Ein **public**-Element ist überall dort sichtbar, wo auf seine Klasse verwiesen werden kann.

Sichtbarkeitsattribute "strict"

Neben den Sichtbarkeitsattributen **private** und **protected** unterstützt der Delphi-Compiler zusätzliche Sichtbarkeitsattribute mit größeren Zugriffsbeschränkungen: **strict private** und **strict protected**. Diese Einstellungen sind mit der .NET Common Language Specification (CLS) konform und können auch in Win32-Anwendungen verwendet werden.

Auf Klassenelemente mit dem Attribut **strict private** kann nur in der Klasse zugegriffen werden, in der sie deklariert sind. Für Prozeduren und Funktionen in derselben Unit sind sie nicht sichtbar. Klassenelemente mit dem Attribut **strict protected** sind nicht nur in der Klasse sichtbar, in der sie deklariert sind, sondern auch in jeder abgeleiteten Klasse. Dabei spielt es keine Rolle,

wo diese deklariert ist. Wenn Instanzelemente (solche, die ohne die Schlüsselwörter **class** oder **class var** deklariert sind) als **strict private** oder **strict protected** deklariert sind, kann auf sie nicht außerhalb der Instanz einer Klasse, in der sie erscheinen, zugegriffen werden. Eine Instanz einer Klasse kann nicht auf **strict protected**- oder **strict protected**-Instanzelemente in anderen Instanzen derselben Klasse zugreifen.

Dem herkömmlichen Delphi-Attribut **private** entspricht die CLR-Sichtbarkeit **assembly**. Das Delphi-Attribut **protected** entspricht der CLR-Sichtbarkeit **assembly** oder **family**.

Anmerkung: Das Wort **strict** wird als Direktive im Kontext einer Klassendeklaration betrachtet. Es ist nicht zulässig, das Wort **strict** innerhalb einer Klassendeklaration als Elementname zu verwenden. Außerhalb der Klassendeklaration kann es jedoch eingesetzt werden.

4

published-Elemente

published-Elemente haben dieselbe Sichtbarkeit wie **public**-Elemente. Im Unterschied zu diesen werden jedoch für **published**-Elemente Laufzeit-Typinformationen generiert. Sie ermöglichen einer Anwendung, die Felder und Eigenschaften eines Objekts dynamisch abzufragen und seine Methoden zu lokalisieren. Diese Informationen werden verwendet, um beim Speichern und Laden von Formulardateien auf die Werte von Eigenschaften zuzugreifen, Eigenschaften im Objektinspektor anzuzeigen und spezielle Methoden (so genannte Ereignisbehandlungsroutine) bestimmten Eigenschaften (den Ereignissen) zuzuordnen.

published-Eigenschaften sind auf bestimmte Datentypen beschränkt. Ordinal-, String-, Klassen-, Interface-, Variant- und Methodenzeigentypen können mit dieser Sichtbarkeit deklariert werden. Bei Mengentypen ist dies nur möglich, wenn die Ober- und Untergrenze des Basistyps einen Ordinalwert zwischen 0 und 31 hat. (Die Menge muss also in ein Byte, Wort oder Doppelwort passen.) Auch alle Real-Typen außer **Real48** können als **published** deklariert werden. Für Eigenschaften von Array-Typen ist **published** dagegen nicht zulässig (im Gegensatz zu den weiter unten beschriebenen Array-Eigenschaften).

Einige Eigenschaften, die als **published** deklariert werden können, werden vom Streaming-System nicht voll unterstützt. Dazu gehören die Eigenschaften von Record-Typen, die Array-Eigenschaften (siehe Seite 965) aller als **published** deklarierbaren Typen und die Eigenschaften von Aufzählungstypen (siehe Seite 889), die anonyme Werte enthalten. Wenn Sie eine derartige Eigenschaft als **published** deklarieren, wird sie im Objektinspektor nicht korrekt angezeigt, und ihr Wert geht beim Schreiben des Streams auf die Festplatte verloren.

Obwohl alle Methoden als **published** deklariert werden können, sind in einer Klasse nicht zwei oder mehr überladene **published**-Methoden mit demselben Namen erlaubt. Felder können nur mit dieser Sichtbarkeit angegeben werden, wenn sie einen Klassen- oder Interface-Typ haben.

Eine Klasse kann nur **published**-Elemente haben, wenn sie mit **{\$M+}** kompiliert wird oder von einer Klasse abgeleitet ist, die mit **{\$M+}** kompiliert wurde. Die meisten Klassen mit **published**-Elementen sind von der Klasse **TPersistent** abgeleitet, die im Status **{\$M+}** kompiliert ist. Die Anweisung **\$M** wird daher nur äußerst selten benötigt.

Anmerkung: Bezeichner, die Unicode-Zeichen enthalten, sind in **published**-Abschnitten von Klassen oder in von **published**-Elementen verwendeten Typen nicht zulässig.

automated-Elemente (nur Win32)

automated-Elemente haben dieselbe Sichtbarkeit wie **public**-Elemente. Im Gegensatz zu diesen werden aber für **automated**-Elemente die für Automatisierungsserver benötigten Automatisierungs-Typinformationen erzeugt. **automated**-Elemente werden normalerweise nur in Win32-Klassen eingesetzt. Vom .NET-Compiler wird das reservierte Wort **automated** nicht akzeptiert. Das reservierte Wort **automated** ist aus Gründen der Abwärtskompatibilität vorhanden. Die Klasse **TAutoObject** der Unit **ComObj** verwendet **automated** nicht.

Für **automated**-Methoden und -Eigenschaften gelten folgende Einschränkungen:

- Die Typen aller Eigenschaften, Array-Eigenschaftsparameter, Methodenparameter und Funktionsergebnisse müssen automatisierbar sein. Dazu gehören die Typen **Byte**, **Currency**, **Real**, **Double**, **Longint**, **Integer**, **Single**, **Smallint**, **AnsiString**, **WideString**, **TDateTime**, **Variant**, **OleVariant** und **WordBool** sowie alle Interface-Typen.
- Methodendeklarationen müssen die Standardaufrufkonvention **register** verwenden. Sie können virtuell sein, nicht aber dynamisch.
- Eigenschaftsdeklarationen dürfen nur die Zugriffsbezeichner (**read** und **write**), aber keine anderen Bezeichner (**index**, **stored**, **default** und **nodefault**) enthalten. Diese Zugriffsbezeichner müssen einen Methodenbezeichner angeben, der die Standardaufrufkonvention **register** verwendet. Feldbezeichner sind nicht zulässig.
- Eigenschaftsdeklarationen müssen einen Typ angeben. Überschreiben ist hier nicht erlaubt.

Bei der Deklaration einer Methode oder Eigenschaft im **automated**-Abschnitt kann optional die Anweisung **dispid** angegeben werden. Die Angabe einer bereits vergebenen Nummer in einer **dispid**-Anweisung führt zu einem Fehler.

Unter Win32 muss auf diese Anweisung eine Integer-Konstante folgen, die die Dispatch-ID für die Automatisierung des Elements angibt. Fehlt diese Angabe, weist der Compiler automatisch eine Dispatch-ID zu. Diese ID ist um 1 höher als die größte Dispatch-ID, die bereits von einer Methode oder Eigenschaft der Klasse bzw. ihrer Vorfahren verwendet wird. Weitere Informationen über die Automatisierung (nur Win32) finden Sie unter Automatisierungsobjekte (siehe Seite 1015).

Vorwärtsdeklarationen und voneinander abhängige Klassen

Wenn die Deklaration eines Klassentyps wie im folgenden Beispiel mit dem Wort **class** und einem Semikolon endet und darauf kein Vorfahr und keine Elemente folgen, handelt es sich um eine Vorwärtsdeklaration:

```
type Klassenname =class;
```

Eine Vorwärtsdeklaration muss durch eine definierende Deklaration dieser Klasse im selben Typdeklarationsabschnitt aufgelöst werden. Zwischen einer Vorwärtsdeklaration und der zugehörigen definierenden Deklaration dürfen also mit Ausnahme anderer Typdeklarationen keine weiteren Anweisungen stehen. Das bedeutet, dass zwischen einer Vorwärtsdeklaration und ihrer definierenden Deklaration ausschließlich andere Typdeklarationen stehen dürfen.

Vorwärtsdeklarationen ermöglichen voneinander abhängige Klassen. Zum Beispiel:

```
type
  TFigure = class; // Vorwärtsdeklaration
  TDrawing = class
    Figure: TFigure;
  ...
end;

TFigure = class // Definierende Deklaration
  Drawing: TDrawing;
  ...
end;
```

Verwechseln Sie Vorwärtsdeklarationen nicht mit vollständigen Deklarationen von Typen, die ohne Angabe von Elementen von **TObject** abgeleitet werden.

```
type
  TFirstClass = class; // Dies ist eine Vorwärtsdeklaration
  TSecondClass = class // Dies ist eine vollständige Klassendeklaration
  end; // 
  TThirdClass = class(TObject); // Dies ist eine vollständige Klassendeklaration
```

Siehe auch

Felder (siehe Seite 954)

Methoden (siehe Seite 956)

Eigenschaften (siehe Seite 965)

Ereignisse (siehe Seite 971)
 Klassenreferenzen (siehe Seite 974)
 Exceptions (siehe Seite 977)
 Verschachtelte Typdeklarationen (siehe Seite 982)
 Überladene Operatoren (siehe Seite 984)
 Unterstützende Klassen (siehe Seite 987)

4.1.1.6.2 Felder

Dieses Thema befasst sich mit der Syntax für die Deklaration von Klassendatenfeldern.

Allgemeines zu Feldern

Ein Feld ist eine Variable, die zu einem bestimmten Objekt gehört. Felder können jeden Typ annehmen, auch Klassentypen (können also Objektreferenzen aufnehmen). Sie werden normalerweise als **private** deklariert.

Um ein Feld als Element einer Klasse zu definieren, deklarieren Sie es einfach wie eine normale Variable. Im folgenden Beispiel wird die Klasse `TNumber` deklariert. Ihr einziges Element ist das Integer-Feld `Int` (außerdem erbt sie natürlich die Methoden von `TObject`).

```
type
  TNumber = class
  var
    Int: Integer;
  end;
```

Das Schlüsselwort **var** ist optional. Aber wenn es nicht angegeben wird, müssen die Felddeklarationen vor allen Eigenschafts- und Methodendeklarationen stehen. Nach Eigenschafts- oder Methodendeklarationen können mit dem Schlüsselwort **var** zusätzliche Felddeklarationen eingeführt werden.

Felder werden statisch gebunden (die Feldreferenzen werden also beim Compilieren aufgelöst). Was dies in der Praxis bedeutet, zeigt das folgende Beispiel:

```
type
  TAncestor = class
  Value: Integer;
  end;

  TDescendant = class(TAncestor)
  Value: string;    // Verdeckt das geerbte Value-Feld
  end;

var
  MyObject: TAncestor;

begin
  MyObject := TDescendant.Create;
  MyObject.Value := 'Hello!'      // Fehler
  (MyObject as TDescendant).Value := 'Hello!'    // Funktioniert!
end;
```

MyObject enthält zwar eine Instanz von `TDescendant`, ist aber als `TAncestor` deklariert. Der Compiler interpretiert daher `MyObject.Value` als Verweis auf das in `TAncestor` deklarierte Integer-Feld. Dennoch sind im `TDescendant`-Objekt beide Felder vorhanden, nur ist das geerbte Feld `Value` durch das neue verdeckt. Der Zugriff ist mithilfe einer Typumwandlung weiterhin möglich.

Konstanten (siehe Seite 926)- und typisierte Konstanten (siehe Seite 926)-Deklarationen können in Klassen und in

nicht-anonymen Records mit globalem Gültigkeitsbereich erscheinen. Konstanten und typisierte Konstanten können außerdem in verschachtelten Typdefinitionen (siehe Seite 982) vorkommen. Konstanten und typisierte Konstanten können nur in Klassendefinitionen erscheinen, wenn die Klasse lokal für eine Prozedur definiert ist (z.B. sie dürfen nicht in Records vorkommen, die innerhalb einer Prozedur definiert sind).

Klassenfelder

Klassenfelder sind Datenfelder in einer Klasse, auf die ohne Objektreferenz zugegriffen werden kann (im Gegensatz zu regulären "Instanzfeldern", die oben besprochen wurden). Die in einem Klassenfeld gespeicherten Daten werden von allen Instanzen der Klasse gemeinsam genutzt. Der Zugriff darauf erfolgt über einen Verweis auf die Klasse oder auf eine Variable, die eine Instanz der Klasse repräsentiert.

Mithilfe einer **class var**-Deklaration können Sie einen Block von Klassenfeldern innerhalb einer Klassendeklaration erzeugen. Alle nach **class var** deklarierten Felder haben statische Speicherattribute. Ein **class var**-Block wird durch Folgendes abgeschlossen:

1. Eine weitere **class var** oder **var**-Deklaration
2. Eine Prozedur- oder Funktionsdeklaration (Methodendeklaration) (siehe Seite 956) (einschließlich Klassenprozeduren und Klassenfunktionen)
3. Eine Eigenschaftsdeklaration (siehe Seite 965) (einschließlich Klasseneigenschaften)
4. Eine Konstruktor- oder Destruktor-Deklaration (siehe Seite 956)
5. Ein Sichtbarkeitsattribut (siehe Seite 948) (**public**, **private**, **protected**, **published**, **strict private** und **strict protected**)

Beispiel:

```
type
  TMyClass = class
    public
      class var           // Ein neuer Block statischer Klassenfelder.
        Red: Integer;
        Green: Integer;
        Blue: Integer;
      var               // Ende des class var-Blocks.
        InstanceField: Integer;
    end;
```

Auf die Klassenfelder Red, Green und Blue kann mit dem folgenden Quelltext zugegriffen werden:

```
TMyClass.Red := 1;
TMyClass.Green := 2;
TMyClass.Blue := 3;
```

Auf Klassenfelder kann auch über eine Instanz der Klasse zugegriffen werden. Mit der folgenden Deklaration:

```
var
  myObject: TMyClass;
```

Dieser Quelltext hat denselben Effekt wie die Zuweisungen an Red, Green und Blue weiter oben:

```
myObject.Red := 1;
myObject.Green := 2;
myObject.Blue := 3;
```

Siehe auch

Klassen und Objekte (siehe Seite 948)

Methoden (siehe Seite 956)

Eigenschaften (siehe Seite 965)

Verschachtelte Typdeklarationen (siehe Seite 982)

Klassenreferenzen (siehe Seite 974)

Exceptions (siehe Seite 977)

Überladene Operatoren (siehe Seite 984)

Unterstützende Klassen (siehe Seite 987)

4.1.1.6.3 Methoden

Eine Methode ist eine Prozedur oder Funktion, die zu einer bestimmten Klasse gehört. Daher wird beim Aufruf einer Methode das Objekt (bzw. bei einer Klassenmethode die Klasse) angegeben, mit dem die Operation durchgeführt werden soll. `SomeObject.Free` ruft beispielsweise die Methode `Free` in `SomeObject` auf.

In diesem Thema werden folgende Bereiche erläutert:

- Methodendeklarationen und -implementierungen
- Methodenbindung
- Methoden überladen
- Konstruktoren und Destruktoren
- Botschaftsmethoden

Allgemeines zu Methoden

Methoden werden in der Deklaration einer Klasse als Prozedur- und Funktionsköpfe angegeben (entsprechend einer Vorwärtsdeklaration). Die verschiedenen Methoden müssen dann nach der Klassendefinition im selben Modul durch eine definierende Deklaration implementiert werden. Die Deklaration der Klasse `TMyClass` im folgenden Beispiel enthält eine Methode mit dem Namen `DoSomething`:

```
type
  TMyClass = class(TObject)
  ...
  procedure DoSomething;
  ...
end;
```

Weiter unten im selben Modul wird die definierende Deklaration für `DoSomething` implementiert:

```
procedure TMyClass.DoSomething;
begin
  ...
end;
```

Während eine Klasse im **interface**- oder **implementation**-Abschnitt einer Unit deklariert werden kann, müssen die definierenden Deklarationen ihrer Methoden im **implementation**-Abschnitt stehen.

In der Kopfzeile einer definierenden Deklaration wird der Name der Methode immer mit der Klasse qualifiziert, zu der die Methode gehört. Optional kann auch die Parameterliste aus der Klassendeklaration wiederholt werden, doch müssen in diesem Fall Reihenfolge, Typ und Name der Parameter genau übereinstimmen. Bei einer Funktion muss auch der Rückgabewert identisch sein.

Methodendeklarationen können spezielle Direktiven enthalten, die in anderen Funktionen oder Prozeduren nicht verwendet werden. Direktiven müssen in der Klassendeklaration enthalten sein (nicht in der definierenden Deklaration) und in der folgenden Reihenfolge angegeben werden:

reintroduce; **overload;** *Bindung*; *Aufrufkonvention*; **abstract;** *Warnung*

Hierbei gilt: *Bindung* ist **virtual**, **dynamic** oder **override**, *Aufrufkonvention* ist **register**, **pascal**, **cdecl**, **stdcall** oder **safecall**, und *Warnung* ist **platform**, **deprecated** oder **library**.

inherited

Das reservierte Wort **inherited** ist für die Polymorphie von großer Bedeutung. Es kann in Methodenimplementierungen (mit oder ohne nachfolgendem Bezeichner) angegeben werden.

Folgt auf **inherited** der Name eines Elements, entspricht dies einem normalen Methodenaufruf bzw. einer Referenz auf eine Eigenschaft oder ein Feld. Der einzige Unterschied besteht darin, dass die Suche nach dem referenzierten Element bei dem direkten Vorfahren der Klasse beginnt, zu der die Methode gehört. Wenn Sie beispielsweise

```
inherited Create(...);
```

in der Definition einer Methode angeben, wird die geerbte Methode `Create` aufgerufen.

Die Anweisung **inherited** ohne Bezeichner verweist auf die geerbte Methode mit demselben Namen wie die aufrufende Methode. Handelt es sich bei der aufrufenden Methode um eine Botschaftsbehandlung, verweist diese auf die geerbte Botschaftsbehandlung für dieselbe Botschaft. In diesem Fall übernimmt **inherited** keine expliziten Parameter, sondern der geerbten Methode werden einfach die Parameter der aufrufenden Methode übergeben. Zum Beispiel wird die Anweisung:

```
inherited;
```

häufig in der Implementierung von Konstruktoren verwendet. Der geerbte Konstruktor wird also mit den Parametern aufgerufen, die an die abgeleitete Klasse übergeben wurden.

Self

Der Bezeichner **Self** verweist in der Implementierung einer Methode auf das Objekt, in dem die Methode aufgerufen wird. Das folgende Beispiel zeigt die Methode `Add` der Klasse `TCollection` (in der Unit `Classes`).

```
function TCollection.Add: TCollectionItem;
begin
  Result := FItemClass.Create(Self);
end;
```

`Add` ruft die Methode `Create` der Klasse auf, auf die das Feld `FItemClass` verweist (immer ein Nachkomme von `TCollectionItem`). Da an `TCollectionItem.Create` ein Parameter des Typs `TCollection` übergeben wird, übergibt `Add` die Instanz von `TCollection`, in der die Methode aufgerufen wird. Der folgende Code veranschaulicht dies:

```
var MyCollection: TCollection;
  ...
  MyCollection.Add // MyCollection wird an die Methode TCollectionItem.Create übergeben
```

Self ist in vielen Situationen hilfreich. So kann beispielsweise ein Element, das in einem Klassentyp deklariert ist, in einer Methode dieser Klasse erneut deklariert werden. In diesem Fall kann mit `Self`.Bezeichner auf das Originalelement zugegriffen werden.

Informationen zu **Self** in Klassenmethoden finden Sie unter Klassenreferenzen (siehe Seite 974).

Methodenbindung

Methodenbindungen können statisch (Standard), virtuell oder dynamisch sein. Virtuelle und dynamische Methoden können überschrieben werden, und sie können abstrakt sein. Diese Angaben spielen eine Rolle, wenn eine Variable eines bestimmten Klassentyps eine Instanz einer abgeleiteten Klasse enthält. Sie bestimmen dann, welche Implementierung beim Aufruf der Methode aktiviert wird.

Statische Methoden

Methoden sind standardmäßig statisch. Beim Aufruf bestimmt der deklarierte Typ (also der Typ zur Compilierzeit) der im Aufruf verwendeten Klassen- bzw. Objektvariablen, welche Implementierung aktiviert wird. Die `Draw`-Methoden im folgenden Beispiel sind statisch:

```
type
  TFigure = class
```

```

procedure Draw;
end;

TRectangle = class(TFigure)
  procedure Draw;
end;

```

Ausgehend von diesen Deklarationen zeigt das folgende Beispiel, wie sich Aufrufe statischer Methoden auswirken. Im zweiten Aufruf von `Figure.Draw` referenziert die Variable `Figure` ein Objekte der Klasse `TRectangle`. Es wird jedoch die `Draw`-Implementierung in `TFigure` aufgerufen, weil `Figure` als `TFigure` deklariert ist.

```

var
  Figure: TFigure;
  Rectangle: TRectangle;

begin
  Figure := TFigure.Create;
  Figure.Draw;           // ruft TFigure.Draw auf
  Figure.Destroy;
  Figure := TRectangle.Create;
  Figure.Draw;           // ruft TFigure.Draw auf

  TRectangle(Figure).Draw; // Aufruf von TRectangle.Draw

  Figure.Destroy;
  Rectangle := TRectangle.Create;
  Rectangle.Draw;         // Aufruf von TRectangle.Draw
  Rectangle.Destroy;
end;

```

Virtuelle und dynamische Methoden

Mithilfe der Direktiven **virtual** und **dynamic** können Methoden als virtuell oder dynamisch deklariert werden. Virtuelle und dynamische Methoden können im Gegensatz zu statischen Methoden in abgeleiteten Klassen überschrieben werden. Beim Aufrufen einer überschriebenen Methode bestimmt nicht der deklarierte, sondern der aktuelle Typ (also der Typ zur Laufzeit) der im Aufruf verwendeten Klassen- bzw. Objektvariable, welche Implementierung aktiviert wird.

Um eine Methode zu überschreiben, braucht sie nur mit der Direktive **override** erneut deklariert zu werden. Dabei müssen Reihenfolge und Typ der Parameter sowie der Typ des Rückgabewertes (falls vorhanden) mit der Deklaration in der Vorfahrtsklasse übereinstimmen.

Im folgenden Beispiel wird die in der Klasse `TFigure` deklarierte Methode `Draw` in zwei abgeleiteten Klassen überschrieben:

```

type
  TFigure = class
    procedure Draw; virtual;
  end;

  TRectangle = class(TFigure)
    procedure Draw; override;
  end;

  TEllipse = class(TFigure)
    procedure Draw; override;
  end;

```

Ausgehend von diesen Deklarationen zeigt der folgende Programmcode, wie sich der Aufruf einer virtuellen Methode durch eine Variable auswirkt, deren aktueller Typ zur Laufzeit geändert wird.

```

var
  Figure: TFigure;
begin

```

```

Figure := TRectangle.Create;
Figure.Draw;           // Aufruf von TRectangle.Draw
Figure.Destroy;
Figure := TEllipse.Create;
Figure.Draw;           // Aufruf von TEllipse.Draw
Figure.Destroy;
end;

```

Nur virtuelle und dynamische Methoden können überschrieben werden. Alle Methoden können jedoch überladen werden (siehe "Methoden überladen").

Der Delphi-Compiler unterstützt auch das Konzept einer *finalen* virtuellen Methode. Durch die Anwendung des Schlüsselworts **final** auf eine virtuelle Methode kann verhindert werden, dass diese von einer abgeleiteten Klasse überschrieben wird. Mit diesem Schlüsselwort wird gleichzeitig dokumentiert, auf welche Weise die Klasse verwendet werden soll. Außerdem ermöglicht es dem Compiler eine Optimierung des generierten Codes.

Unterschiede zwischen virtuellen und dynamischen Methoden

In Delphi für .NET sind virtuelle und dynamische Methoden identisch. In Delphi für Win32 sind virtuelle und dynamische Methoden von der Semantik her identisch. Sie unterscheiden sich aber bei der Implementierung der Aufrufverteilung zur Laufzeit. Virtuelle Methoden werden auf Geschwindigkeit, dynamische Methoden auf Code-Größe optimiert.

Im Allgemeinen kann mit virtuellen Methoden polymorphes Verhalten am effizientesten implementiert werden. Dynamische Methoden sind hilfreich, wenn in einer Basisklasse eine große Anzahl überschreibbarer Methoden deklariert ist, die von vielen abgeleiteten Klassen geerbt, aber nur selten überschrieben werden.

Anmerkung: Verwenden Sie dynamische Methoden nur, wenn sich daraus ein nachweisbarer Nutzen ergibt. Allgemein sollten Sie virtuelle Methoden verwenden.

Unterschiede zwischen Überschreiben und Verdecken

Wenn in einer Methodendeklaration dieselben Bezeichner- und Parameterangaben wie bei einer geerbten Methode ohne die Anweisung **override** angegeben werden, wird die geerbte Methode durch die neue Deklaration verdeckt. Beide Methoden sind jedoch in der abgeleiteten Klasse vorhanden, in der die Methode statisch gebunden wird. Zum Beispiel:

```

type
  T1 = class(TObject)
    procedure Act; virtual;
  end;

  T2 = class(T1)
    procedure Act;    // Act wird erneut deklariert, aber nicht überschrieben
  end;

var
  SomeObject: T1;

begin
  SomeObject := T2.Create;
  SomeObject.Act;    // Aufruf von T1.Act
end;

```

reintroduce

Mithilfe der Anweisung **reintroduce** kann verhindert werden, dass der Compiler Warnungen ausgibt, wenn eine zuvor deklarierte virtuelle Methode verdeckt wird. Zum Beispiel:

```
procedure DoSomething; reintroduce; // In der Vorfahrtsklasse existiert ebenfalls eine
```

DoSomething-Methode

Verwenden Sie **reintroduce**, wenn eine geerbte virtuelle Methode durch eine neue Deklaration verdeckt werden soll.

Abstrakte Methoden

Eine abstrakte Methode ist eine virtuelle oder dynamische Methode, die nicht in der Klasse implementiert wird, in der sie deklariert ist. Die Implementierung wird erst später in einer abgeleiteten Klasse durchgeführt. Bei der Deklaration abstrakter Methoden muss die Anweisung **abstract** nach **virtual** oder **dynamic** angegeben werden. Zum Beispiel:

```
procedure DoSomething; virtual; abstract;
```

Eine abstrakte Methode kann nur in einer Klasse (bzw. Instanz einer Klasse) aufgerufen werden, in der sie überschrieben wurde.

Anmerkung: In Delphi für .NET ist es zulässig, eine komplette Klasse als abstrakt zu deklarieren, obwohl keine abstrakten virtuellen Methoden zur Verfügung stehen. Weitere Informationen finden Sie unter [Klassentypen](#) (siehe Seite 948).

Klassenmethoden

Die meisten Methoden werden Instanzmethoden genannt, weil sie mit einer einzelnen Instanz eines Objekts arbeiten. Eine Klassenmethode ist eine Methode, die nicht mit Objekten, sondern mit Klassen arbeitet. Es gibt zwei Typen von Klassenmethoden: reguläre Klassenmethoden und klassenstatische Methoden.

Reguläre Klassenmethoden

Die Definition muss mit dem reservierten Wort **class** beginnen. Zum Beispiel:

```
type
  TFigure = class
  public
    class function Supports(Operation: string): Boolean; virtual;
    class procedure GetInfo(var Info: TFigureInfo); virtual;
    ...
  end;
```

Auch die definierende Deklaration einer Klassenmethode muss mit **class** eingeleitet werden. Zum Beispiel:

```
class procedure TFigure.GetInfo(var Info: TFigureInfo);
begin
  ...
end;
```

In der definierenden Deklaration einer Klassenmethode kann mit dem Bezeichner **Self** auf die Klasse zugegriffen werden, in der die Methode aufgerufen wird (dies kann auch ein Nachkomme der Klasse sein, in der sie definiert ist). Wird die Methode beispielsweise in der Klasse *K* aufgerufen, hat **Self** den Typ *class of K*. Daher können Sie **Self** nicht für den Zugriff auf Instanzfelder, Instanzeigenschaften und normale (Objekt-) Methoden, sondern nur für Aufrufe von Konstruktoren und anderen Klassenmethoden oder für den Zugriff auf Klasseneigenschaften und -feldern verwenden.

Eine Klassenmethode kann über eine Klassenreferenz oder eine Objektreferenz aufgerufen werden. Bei einer Objektreferenz erhält **Self** als Wert die Klasse des betreffenden Objekts.

Klassenstatische Methoden

Auf klassenstatische Methoden kann, wie auf Klassenmethoden, ohne Objektreferenz zugegriffen werden. Im Gegensatz zu regulären Klassenmethoden haben klassenstatische Methoden keinen **Self**-Parameter. Sie können außerdem auf keine Instanzelemente zugreifen. (Sie können aber auf Klassenfelder, Klasseneigenschaften und Klassenmethoden zugreifen.) Wiederum im Gegensatz zu Klassenmethoden können klassenstatische Methoden nicht als **virtual** deklariert werden.

Um eine Methode als klassenstatisch zu deklarieren, fügen Sie das Wort **static** an die Deklaration an:

```
type
  TMyClass = class
```

```

strict private
  class var
    FX : Integer;

strict protected

  // Hinweis: Methoden für den Zugriff auf Klasseneigenschaften müssen als
  // klassenstatisch deklariert werden.
  class function GetX: Integer; static;
  class procedure SetX(val: Integer); static;

public
  class property X: Integer read GetX write SetX;
  class procedure StatProc(s: String); static;
end;

```

Wie eine Klassenmethode kann eine klassenstatische Methode über den Klassentyp (also ohne Objektreferenz) aufgerufen werden, z. B.:

```
TMyClass.X := 17;
TMyClass.StatProc('Hello');
```

Methoden überladen

Eine Methode kann auch mit der Direktive **overload** neu deklariert werden. Wenn sich die Parameterangaben von denen ihres Vorfahren unterscheiden, wird die geerbte Methode überladen, ohne dass sie dadurch verdeckt wird. Bei einem Aufruf der Methode in einer abgeleiteten Klasse wird dann diejenige Implementierung aktiviert, bei der die Parameter übereinstimmen.

Verwenden Sie beim Überladen einer virtuellen Methode die Direktive **reintroduce**, wenn die Methode in einer abgeleiteten Klasse neu deklariert wird. Zum Beispiel:

```

type
  T1 = class(TObject)
    procedure Test(I: Integer); overload; virtual;
  end;

  T2 = class(T1)
    procedure Test(S: string); reintroduce; overload;
  end;
  ...

SomeObject := T2.Create;
SomeObject.Test('Hello!');           // Aufruf von T2.Test
SomeObject.Test(7);                 // Aufruf von T1.Test

```

Innerhalb einer Klasse dürfen nicht mehrere überladene Methoden mit demselben Namen als **published** deklariert werden. Die Pflege von Typinformationen zur Laufzeit bedarf für jedes als **published** deklariertes Element eines eindeutigen Namens.

```

type
  TSomeClass = class
  published
    function Func(P: Integer): Integer;
    function Func(P: Boolean): Integer;    // Fehler
  end;

```

Methoden, die als **read**- oder **write**-Bezeichner für Eigenschaften fungieren, können nicht überladen werden.

Bei der Implementierung einer überladenen Methode muss die Parameterliste aus der Klassendeklaration wiederholt werden. Weitere Informationen zum Überladen finden Sie unter Prozeduren und Funktionen überladen (siehe Seite 931).

Konstruktoren

Ein Konstruktor ist eine spezielle Methode, mit der Instanzobjekte erstellt und initialisiert werden können. Die Deklaration gleicht einer normalen Prozedurdeklaration, beginnt aber mit dem Wort **constructor**. Beispiele:

```
constructor Create;
constructor Create(AOwner: TComponent);
```

Für Konstruktoren muss die Standard-Aufrufkonvention **register** verwendet werden. Obwohl die Deklaration keinen Rückgabewert enthält, gibt ein Konstruktor immer einen Verweis auf das Objekt zurück, das er erstellt bzw. in dem er aufgerufen wird.

Eine Klasse kann auch mehrere Konstruktoren haben. Im Normalfall ist jedoch nur einer vorhanden. Konstruktoren heißen normalerweise immer **Create**.

Das folgende Beispiel zeigt, wie Sie ein Objekt durch einen Aufruf des Konstruktors eines Klassentyps erstellen können: Zum Beispiel:

```
MyObject := TMyClass.Create;
```

Diese Anweisung reserviert zuerst Speicher für das neue Objekt. Anschließend wird allen Ordinalfeldern der Wert Null, allen Zeigern und Klassentypfeldern der Wert **nil** und allen String-Feldern ein leerer String zugewiesen. Danach werden die weiteren Aktionen in der Implementierung des Konstruktors ausgeführt (z. B. Initialisieren der Objekte mit den als Parameter übergebenen Werten). Am Ende gibt der Konstruktor eine Referenz auf das neu erstellte und initialisierte Objekt zurück. Der Typ entspricht dem im Aufruf angegebenen Klassentyp.

Tritt in einem mit einer Klassenreferenz aufgerufenen Konstruktor eine Exception auf, wird das unvollständige Objekt automatisch durch einen Aufruf des Destruktors **Destroy** freigegeben.

Wenn Sie einen Konstruktor mit einer Objektreferenz (anstatt mit einer Klassenreferenz) aufrufen, wird kein Objekt erstellt. Stattdessen werden wie bei einer normalen Routine die angegebenen Anweisungen mit dem Objekt ausgeführt, und es wird eine Referenz auf das Objekt zurückgegeben. Beim Aufruf mit einer Objektreferenz wird normalerweise der geerbte Konstruktor mit **inherited** ausgeführt.

Das folgende Beispiel zeigt einen Klassentyp und den zugehörigen Konstruktor:

```
type
  TShape = class(TGraphicControl)
  private
    FPen: TPen;
    FBrush: TBrush;
    procedure PenChanged(Sender: TObject);
    procedure BrushChanged(Sender: TObject);
  public
    constructor Create(Owner: TComponent); override;
    destructor Destroy; override;
    ...
  end;

constructor TShape.Create(Owner: TComponent);
begin
  inherited Create(Owner);      // Initialisieren geerbter Komponenten
  Width := 65;                // Ändern geerbter Eigenschaften
  Height := 65;
  FPen := TPen.Create; // Initialisieren neuer Felder
  FPen.OnChange := PenChanged;
  FBrush := TBrush.Create;
  FBrush.OnChange := BrushChanged;
end;
```

Als erste Anweisung wird normalerweise immer der geerbte Konstruktor aufgerufen, um die geerbten Felder zu initialisieren. Danach werden den in der abgeleiteten Klasse deklarierten Feldern Werte zugewiesen. Da der Konstruktor grundsätzlich den Speicherbereich bereinigt, der dem neuen Objekt zugewiesen wird, erhalten alle Felder automatisch den Anfangswert Null (Ordinaltypen), **nil** (Zeiger und Klassentypen), Leerstring (String-Typen) oder Unassigned (Varianten). Aus diesem Grund brauchen nur solche Felder explizit initialisiert zu werden, denen ein Anfangswert ungleich Null (bzw. kein Leerstring) zugewiesen werden soll.

Ein als **virtual** deklarierter Konstruktor, der mit einem Klassentypbezeichner aufgerufen wird, entspricht einem statischen Konstruktor. In Verbindung mit Klassenreferenztypen können jedoch durch virtuelle Konstruktoren Objekte polymorph erstellt werden (d. h. der Objekttyp ist beim Compilieren noch nicht bekannt). Weitere Informationen finden Sie unter Klassenreferenzen

([siehe Seite 974](#)).

Anmerkung: Informationen zu Konstruktoren, Destruktoren und der Speicherverwaltung auf der .NET-Plattform finden Sie im Thema Speicherverwaltung auf der .NET-Plattform ([siehe Seite 1027](#)).

Der Klassenkonstruktor (.NET)

Klassenkonstruktoren werden ausgeführt, bevor eine Klasse referenziert oder verwendet wird. Der Klassenkonstruktor muss als **strict private** deklariert werden. In jeder Klasse darf nur ein Klassenkonstruktor vorhanden sein. Nachkommen können zwar ihren eigenen Klassenkonstruktor deklarieren, rufen aber im Rumpf des Klassenkonstruktors nicht **inherited** auf. Es ist nicht möglich, einen Klassenkonstruktor direkt aufzurufen oder auf ihn zuzugreifen (etwa über seine Adresse), da der Code für seinen Aufruf vom Compiler erzeugt wird.

Der Klassenkonstruktor wird vor der Verwendung der Klasse ausgeführt. Der genaue Aufrufzeitpunkt kann jedoch nicht exakt vorherbestimmt werden. Damit eine Klasse auf der .NET-Plattform verwendet werden kann, muss sie sich im ausgeführten Code befinden. Wenn eine Klasse z. B. erstmalig in einer **if**-Anweisung referenziert wird und während der Ausführung der Test der **if**-Anweisung nie **True** ergibt, wird die Klasse nie geladen und vom JIT-Compiler nicht berücksichtigt. Der Klassenkonstruktor wird deshalb nicht aufgerufen.

Die folgende Klassendeklaration veranschaulicht die Syntax von Klasseneigenschaften und -feldern, klassenstatistischen Methoden und Klassenkonstruktoren:

```
type
  TMyClass = class
    strict protected
      // Methoden für den Zugriff auf Klasseneigenschaften müssen als klassenstatisch
      deklariert werden.
      class function GetX: Integer; static;
      class procedure SetX(val: Integer); static;
    public
      class property X: Integer read GetX write SetX;
      class procedure StatProc(s: String); static;
    strict private
      class var
        FX : Integer;
      class constructor Create;
    end;
```

Destruktoren

Ein Destruktor ist eine spezielle Methode, die ein Objekt im Speicher freigibt. Die Deklaration gleicht einer normalen Prozedurdeklaration, beginnt aber mit dem Wort **destructor**. Ein Beispiel:

```
destructor SpecialDestructor(SaveData: Boolean);
destructor Destroy; override;
```

In Win32 muss für Destruktoren die Standard-Aufrufkonvention **register** verwendet werden. Obwohl in einer Klasse mehrere Destruktoren implementiert werden können, ist es ratsam, nur die geerbte Methode **Destroy** zu überschreiben und keine weiteren Destruktoren zu deklarieren.

Ein Destruktor kann nur über ein Instanzobjekt aufgerufen werden. Zum Beispiel:

```
MyObject.Destroy;
```

Beim Aufruf eines Destruktors werden zuerst die in der Implementierung angegebenen Aktionen ausgeführt. Normalerweise werden hier untergeordnete Objekte und zugewiesene Ressourcen freigegeben. Danach wird der durch das Objekt belegte Speicherplatz freigegeben.

Das folgende Beispiel zeigt eine typische Destruktorimplementierung:

```
destructor TShape.Destroy;
begin
```

```

FBrush.Free;
FPen.Free;
inherited Destroy;
end;

```

Die letzte Anweisung ruft den geerbten Destruktor auf, der die geerbten Felder freigibt.

Wenn beim Erstellen eines Objekts eine Exception auftritt, wird das unvollständige Objekt automatisch durch einen Aufruf von `Destroy` freigegeben. Der Destruktor muss daher auch in der Lage sein, Objekte freizugeben, die nur teilweise erstellt wurden. Da im Konstruktor alle Felder eines neuen Objekts mit Null initialisiert werden, haben Klassenreferenz- und Zeigerfelder in einer unvollständigen Instanz immer den Wert `nil`. Testen Sie solche Felder im Destruktor immer auf den Wert `nil`, bevor Sie Operationen mit ihnen durchführen. Wenn Sie Objekte nicht mit `Destroy`, sondern mit der Methode `Free` (von `TObject`) freigeben, wird diese Prüfung automatisch durchgeführt.

Anmerkung: Informationen zu Konstruktoren, Destruktoren und der Speicherverwaltung auf der .NET-Plattform finden Sie im Thema Speicherverwaltung auf der .NET-Plattform (siehe Seite 1027).

Botschaftsmethoden

In Botschaftsmethoden können Reaktionen auf dynamisch gesendete Botschaften implementiert werden. Die Syntax für Botschaftsmethoden wird auf allen Plattformen unterstützt. In der VCL werden Botschaftsmethoden verwendet, um auf Windows-Botschaften zu antworten.

Sie erstellen eine Botschaftsmethode, indem Sie die Direktive `message` und eine Integer-Konstante zwischen 1 und 49151 (die so genannte Botschafts-ID) in eine Methodendeklaration aufnehmen. In Botschaftsmethoden für VCL-Steuerelemente kann als Integer-Konstante eine der Botschafts-IDs von Win32 verwendet werden, die (zusammen mit den entsprechenden Record-Typen) in der Unit `Messages` definiert sind. Eine Botschaftsmethode muss immer eine Prozedur mit einem einzelnen `var`-Parameter sein.

Ein Beispiel:

```

type
  TTextBox = class(TCustomControl)
  private
    procedure WMChar(var Message: TWMChar); message WM_CHAR;
  ...
end;

```

In einer Botschaftsmethode braucht die Direktive `override` nicht angegeben zu werden, um eine geerbte Botschaftsmethode zu überschreiben. Es muss nicht einmal derselbe Methodenname oder Parametertyp wie bei der zu überschreibenden Methode verwendet werden. Allein die Botschafts-ID bestimmt, auf welche Botschaft die Methode reagiert, und ob es sich um eine überschriebene Methode handelt.

Botschaftsmethoden implementieren

In der Implementierung einer Botschaftsmethode kann die geerbte Botschaftsmethode wie im folgenden Beispiel aufgerufen werden:

```

procedure TTextBox.WMChar(var Message: TWMChar);
begin
  if MessageCharCode = Ord(#13) then
    ProcessEnter
  else
    inherited;
end;

```

Die Anweisung `inherited` durchsucht die Klassenhierarchie nach oben und ruft die erste Botschaftsmethode mit derselben ID wie die aktuelle Methode auf. Dabei wird automatisch der Botschafts-Record übergeben. Ist in keiner Vorfahrklasse eine Botschaftsmethode mit dieser ID implementiert, wird die in `TObject` definierte Originalmethode `DefaultHandler` aufgerufen.

Die Implementierung von `DefaultHandler` in `TObject` gibt einfach die Steuerung zurück, ohne eine Aktion auszuführen. Durch

Überschreiben von `DefaultHandler` kann in einer Klasse eine eigene Standardbehandlung implementiert werden. In Win32 ruft die `DefaultHandler`-Methode für Steuerelemente die `DefWindowProc`-Funktion der Win32-API auf.

Botschaftsweiterleitung

Botschaftsmethoden werden normalerweise nicht direkt aufgerufen. Stattdessen werden die Botschaften mithilfe der von `TObject` geerbten Methode `Dispatch` an ein Objekt weitergeleitet:

```
procedure Dispatch(var Message);
```

Der Parameter `Message` muss ein Record sein und als erstes Element ein **Word**-Feld mit einer Botschafts-ID enthalten.

`Dispatch` durchsucht die Klassenhierarchie nach oben (beginnend bei der Klasse des Objekts, in dem sie aufgerufen wird) und ruft die erste für die übergebene ID gefundene Botschaftsmethode auf. Wird keine solche Methode gefunden, ruft `Dispatch` die Methode `DefaultHandler` auf.

Siehe auch

Klassen und Objekte ([siehe Seite 948](#))

Felder ([siehe Seite 954](#))

Eigenschaften ([siehe Seite 965](#))

Verschachtelte Typdeklarationen ([siehe Seite 982](#))

Klassenreferenzen ([siehe Seite 974](#))

Exceptions ([siehe Seite 977](#))

Überladene Operatoren ([siehe Seite 984](#))

Unterstützende Klassen ([siehe Seite 987](#))

4.1.1.6.4 Eigenschaften

Dieses Thema enthält Informationen zu folgenden Bereichen:

- Auf Eigenschaften zugreifen
- Array-Eigenschaften
- Indexbezeichner
- Speicherbezeichner
- Eigenschaften überschreiben und neu deklarieren
- Klasseneigenschaften

Allgemeines zu Eigenschaften

Eine Eigenschaft definiert wie ein Feld ein Objektattribut. Felder sind jedoch nur Speicherbereiche, die überprüft und geändert werden können, während Eigenschaften mithilfe bestimmter Aktionen gelesen und geschrieben werden können. Sie erlauben eine größere Kontrolle über den Zugriff auf die Attribute eines Objekts und ermöglichen das Berechnen von Attributen.

Die Deklaration einer Eigenschaft muss einen Namen, einen Typ und mindestens eine Zugriffsangabe enthalten. Die Syntax lautet folgendermaßen:

property Eigenschaftsname[Indizes]: Typindex Integer-Konstante Bezeichner;

Die Variablen haben folgende Bedeutung:

- *Eigenschaftsname* ist ein beliebiger gültiger Bezeichner.
- *[Indizes]* ist optional und besteht aus einer Folge von durch Strichpunkte getrennten Parameterdeklarationen. Jede

Deklaration hat die Form *Bezeichner1*, ..., *Bezeichnern*: Typ. Weitere Informationen finden Sie im Abschnitt "Array-Eigenschaften" weiter unten.

- Typ muss ein vordefinierter oder vorher deklarierter Typbezeichner sein. Eigenschaftsdeklarationen der Form property Num: 0..9 sind somit unzulässig.
- Die Indexklausel *Integer-Konstante* ist optional. Weitere Informationen hierzu finden Sie im Abschnitt "Indexbezeichner" weiter unten.
- Bezeichner besteht aus den Bezeichnern **read**, **write**, **stored**, **default** (oder **nodefault**) und **implements**. Jede Eigenschaftsdeklaration muss zumindest einen **read**- oder **write**-Bezeichner enthalten.

Eigenschaften werden durch ihre Zugriffsbezeichner definiert. Sie können im Gegensatz zu Feldern nicht als **var**-Parameter übergeben oder mit dem Adressoperator @ versehen werden. Der Grund dafür liegt darin, dass eine Eigenschaft nicht notwendigerweise im Speicher vorhanden sein muss. Sie kann beispielsweise eine **read**-Methode haben, die einen Wert aus einer Datenbank abruft oder einen Zufallswert generiert.

Auf Eigenschaften zugreifen

Jede Eigenschaft verfügt über eine **read**- oder eine **write**-Angabe (oder über beide). Diese Zugriffsbezeichner werden in folgender Form angegeben:

read *FeldOderMethode*

write *FeldOderMethode*

FeldOderMethode ist der Name eines Feldes oder einer Methode, das bzw. die in derselben Klasse oder in einer Vorfahrklasse der Eigenschaft deklariert ist.

- Wenn *FeldOderMethode* in derselben Klasse deklariert wird, muss dies vor der Eigenschaft geschehen. Ist das Element in einer Vorfahrklasse deklariert, muss es in der abgeleiteten Klasse sichtbar sein. Es kann also kein Element einer abgeleiteten Klasse angegeben werden, das in einer anderen Unit als **private** deklariert ist.
- Handelt es sich bei *FeldOderMethode* um ein Feld, muss dieses Feld denselben Typ wie die Eigenschaft haben.
- Wenn *FeldOderMethode* eine Methode ist, kann diese nicht **dynamisch** sein. Falls es sich um eine **virtuelle** Methode handelt, kann diese nicht überladen werden. Zugriffsmethoden für eine als **published** deklarierte Eigenschaft müssen die Standard-Aufrufkonvention **register** verwenden.
- Wird in einem **read**-Bezeichner für **FeldOderMethode** eine Methode angegeben, muss eine Funktion ohne Parameter verwendet werden, die einen Wert mit dem Typ der Eigenschaft zurückgibt. Eine Ausnahme bildet die Zugriffsmethode für eine indizierte Eigenschaft oder eine Array-Eigenschaft.
- Wird in einem **write**-Bezeichner für *FeldOderMethode* eine Methode angegeben, muss eine Prozedur verwendet werden, die einen Wert- oder **const**-Parameter mit dem Datentyp der Eigenschaft entgegennimmt (oder wenn es sich um eine Array-Eigenschaft oder indizierte Eigenschaft handelt).

Betrachten Sie beispielsweise die folgende Deklaration:

```
property Color: TColor read GetColor write SetColor;
```

Die Methode *GetColor* muss hier folgendermaßen deklariert werden:

```
function GetColor: TColor;
```

Die Deklaration der Methode *SetColor* muss eine der folgenden Formen haben:

```
procedure SetColor(Value: TColor);
procedure SetColor(const Value: TColor);
```

(Der Parameter von *SetColor* muss natürlich nicht unbedingt *Value* heißen.)

Wenn eine Eigenschaft in einem Ausdruck verwendet wird, erfolgt der Zugriff auf ihren Wert mithilfe des mit **read** angegebenen Elements (Feld oder Methode). Bei Zuweisungen wird das mit **write** angegebene Element für das Schreiben der Eigenschaft verwendet.

Im folgenden Beispiel wird die Klasse *TCompass* mit der **published**-Eigenschaft *Heading* deklariert. Zum Lesen der

Eigenschaft wird das Feld `FHeading`, zum Schreiben die Prozedur `SetHeading` verwendet.

```
type
  THeading = 0..359;
  TCompass = class(TControl)
  private
    FHeading: THeading;
    procedure SetHeading(Value: THeading);
  published
    property Heading: THeading read FHeading write SetHeading;
  ...
end;
```

Ausgehend von dieser Deklaration ist die Anweisung

```
if Compass.Heading = 180 then GoingSouth;
Compass.Heading := 135;
```

mit den folgenden Anweisungen identisch:

```
if Compass.FHeading = 180 then GoingSouth;
Compass.SetHeading(135);
```

In der Klasse `TCompass` wird zum Lesen der Eigenschaft `Heading` keine bestimmte Aktion verwendet. Die `read`-Operation besteht lediglich aus dem Abrufen des im Feld `FHeading` gespeicherten Wertes. Wertzuweisungen an die Eigenschaft werden in Aufrufe der Methode `SetHeading` umgesetzt. Diese Methode speichert den neuen Wert im Feld `FHeading` und führt optional weitere Operationen durch. `SetHeading` könnte beispielsweise folgendermaßen implementiert werden:

```
procedure TCompass.SetHeading(Value: THeading);
begin
  if FHeading <> Value then
    begin
      FHeading := Value;
      Repaint; // Aktualisieren der Benutzeroberfläche, um den neuen Wert
      anzeigen
    end;
end;
```

Eine Eigenschaft, die nur mit einer `read`-Angabe deklariert ist, nennt man Nur-Lesen-Eigenschaft. Ist nur ein `write`-Bezeichner vorhanden, handelt es sich um eine Nur-Schreiben-Eigenschaft. Wenn Sie einer Nur-Lesen-Eigenschaft einen Wert zuweisen oder eine Nur-Schreiben-Eigenschaft in einem Ausdruck verwenden, tritt ein Fehler auf.

Array-Eigenschaften

Array-Eigenschaften sind indizierte Eigenschaften. Sie werden beispielsweise für die Einträge eines Listenfeldes, die untergeordneten Objekte eines Steuerelements oder die Pixel einer Bitmap-Grafik verwendet.

Die Deklaration einer Array-Eigenschaft enthält eine Parameterliste mit den Namen und Typen der Indizes. Zum Beispiel:

```
property Objects[Index: Integer]: TObject read GetObject write SetObject;
property Pixels[X, Y: Integer]: TColor read GetPixel write SetPixel;
property Values[const Name: string]: string read GetValue write SetValue;
```

Das Format der Liste ist mit dem Format der Parameterliste einer Prozedur oder Funktion identisch. Der einzige Unterschied besteht darin, dass die Parameterdeklarationen nicht in runden, sondern in eckigen Klammern angegeben werden. Im Gegensatz zu Arrays, bei denen nur ordinale Indizes erlaubt sind, können die Indizes von Array-Eigenschaften einen beliebigen Typ haben.

Bei Array-Eigenschaften müssen die Zugriffsbezeichner keine Felder, sondern Methoden angeben. Die Methode in einem `read`-Bezeichner muss eine Funktion sein, bei der Anzahl, Reihenfolge und Typ der Parameter mit der Indexparameterliste der Eigenschaft identisch sind und der Ergebnistyp mit dem Typ der Eigenschaft übereinstimmt. Die Methode in einem `write`-Bezeichner muss eine Prozedur sein, bei der Anzahl, Reihenfolge und Typ der Parameter mit der Indexparameterliste der Eigenschaft identisch sind. Außerdem muss ein zusätzlicher Wert- oder `const`-Parameter mit dem Typ der Eigenschaft vorhanden sein.

Die Zugriffsmethoden für die obigen Array-Eigenschaften können beispielsweise folgendermaßen deklariert werden:

```
function GetObject(Index: Integer): TObject;
function GetPixel(X, Y: Integer): TColor;
function GetValue(const Name: string): string;
procedure SetObject(Index: Integer; Value: TObject);
procedure SetPixel(X, Y: Integer; Value: TColor);
procedure SetValue(const Name, Value: string);
```

Auf eine Array-Eigenschaft kann durch Indizieren ihres Bezeichners zugegriffen werden. So sind beispielsweise die Anweisungen

```
if Collection.Objects[0] = nil then Exit;
Canvas.Pixels[10, 20] := clRed;
Params.Values['PATH'] := 'C:\BIN';
```

mit den folgenden Anweisungen identisch:

```
if Collection.GetObject(0) = nil then Exit;
Canvas.SetPixel(10, 20, clRed);
Params.SetValue('PATH', 'C:\BIN');
```

Wenn Sie die Direktive **default** nach der Definition einer Array-Eigenschaft angeben, wird diese als Standardeigenschaft der betreffenden Klasse verwendet. Zum Beispiel:

```
type
  TStringArray = class
  public
    property Strings[Index: Integer]: string ...; default;
  end;
```

Auf die Array-Standardeigenschaft einer Klasse kann mit der Kurzform `Objekt[Index]` zugegriffen werden. Diese Anweisung ist mit `Objekt.Eigenschaft[Index]` identisch. Ausgehend von der vorhergehenden Deklaration kann z. B. `StringArray.Strings[7]` zu `StringArray[7]` verkürzt werden. Eine Klasse kann nur eine Standardeigenschaft mit der gegebenen Signatur (Array-Parameterliste) haben, aber es ist möglich, die Standardeigenschaft zu überladen. Das Ändern oder Verbergen der Standardeigenschaft in abgeleiteten Klassen kann zu unerwünschten Ergebnissen führen, da der Compiler Bindungen an Eigenschaften immer statisch vornimmt.

Indexbezeichner

Mithilfe von Indexbezeichnern können mehrere Eigenschaften dieselbe Zugriffsmethode verwenden, auch wenn sie unterschiedliche Werte repräsentieren. Indexbezeichner bestehen aus der Direktive **index** und einem Integer-Wert zwischen -2.147.483.647 und 2.147.483.647. Bei Eigenschaften mit Indexbezeichnern muss auf die Direktiven **read** und **write** eine Methode (kein Feld) folgen. Zum Beispiel:

```
type
  TRectangle = class
  private
    FCoordinates: array[0..3] of Longint;
    function GetCoordinate(Index: Integer): Longint;
    procedure SetCoordinate(Index: Integer; Value: Longint);
  public
    property Left: Longint index 0 read GetCoordinate write SetCoordinate;
    property Top: Longint index 1 read GetCoordinate write SetCoordinate;
    property Right: Longint index 2 read GetCoordinate write SetCoordinate;
    property Bottom: Longint index 3 read GetCoordinate write SetCoordinate;
    property Coordinates[Index: Integer]: Longint read GetCoordinate write SetCoordinate;
  end;
```

Eine Zugriffsmethode für eine Eigenschaft mit einem Indexbezeichner benötigt einen zusätzlichen Wert-Parameter vom Typ Integer. Bei einer **read**-Funktion muss dies der letzte, bei einer **write**-Prozedur der vorletzte Parameter sein. Diese Konstante (der Index) wird beim Zugriff auf die Eigenschaft automatisch an die Zugriffsmethode übergeben.

Wenn Rectangle ein Objekt der zuvor deklarierten Klasse `TRectangle` ist, dann ist die Anweisung

```
Rectangle.Right := Rectangle.Left + 100;
```

mit der folgenden Anweisung identisch:

```
Rectangle.SetCoordinate(2, Rectangle.GetCoordinate(0) + 100);
```

Speicherbezeichner

Die optionalen Direktiven **stored**, **default** und **nodefault** sind Speicherbezeichner. Sie haben keinerlei Auswirkungen auf die Funktionsweise des Programms, sondern bestimmen, ob die Werte der **published**-Eigenschaften in Formulardateien gespeichert werden.

Nach der Angabe **stored** muss der Wert **True** oder **False**, der Name eines Booleschen Feldes oder der Name einer parameterlosen Methode folgen, die einen Booleschen Wert zurückgibt. Zum Beispiel:

```
property Name : TComponentName read FName write SetName stored False;
```

Wird eine Eigenschaft ohne die Angabe von **stored** deklariert, entspricht dies der Definition **stored True**.

Nach **default** muss eine Konstante angegeben werden, die denselben Datentyp wie die Eigenschaft hat. Zum Beispiel:

```
property Tag: Longint read FTag write FTag default 0;
```

Mithilfe des Bezeichners **nodefault** kann ein geerbter **default**-Wert ohne Angabe eines neuen Wertes außer Kraft gesetzt werden. Die Direktiven **default** und **nodefault** werden nur für Ordinal- und Mengentypen unterstützt, bei denen die Ober- und Untergrenze des Basistyps einen Ordinalwert zwischen 0 und 31 hat. Enthält eine Eigenschaftsdeklaration weder **default** noch **nodefault**, gilt sie als mit **nodefault** definiert. Für Real-, Zeiger- und String-Typen gilt der implizite **default**-Wert 0 bzw. **nil** und **''** (leerer String).

Anmerkung: Sie können den ordinalen Wert 2147483648 nicht als Standardwert verwenden. Dieser Wert wird intern für die Darstellung von **nodefault** verwendet.

Beim Speichern einer Komponente werden die Speicherbezeichner ihrer **published**-Eigenschaften überprüft. Wenn sich der aktuelle Wert einer Eigenschaft von ihrem **default**-Wert unterscheidet (oder kein **default**-Wert vorhanden ist) und **stored True** ist, wird der Wert gespeichert. Treffen diese Bedingungen nicht zu, wird der Wert nicht gespeichert.

Anmerkung: Eigenschaftswerte werden nicht automatisch auf den Standardwert initialisiert, d. h. die Standarddirektive steuert nur das Speichern von Eigenschaftswerten in der Formulardatei und nicht das Speichern des ersten Werts der Eigenschaft in einer neu erstellten Instanz.

Bei Array-Eigenschaften werden Speicherbezeichner nicht unterstützt. Bei diesem Eigenschaftstyp hat **default** eine andere Bedeutung (siehe "Array-Eigenschaften" weiter oben).

Eigenschaften überschreiben und neu deklarieren

Das Deklarieren einer Eigenschaft ohne Angabe eines Typs nennt man Überschreiben. Diese Vorgehensweise ermöglicht das Ändern der geerbten Sichtbarkeit bzw. des geerbten Bezeichners einer Eigenschaft. In der einfachsten Form braucht nur das reservierte Wort **property** zusammen mit einem geerbten Eigenschaftsbezeichner angegeben zu werden. Auf diese Weise kann die Sichtbarkeit der betreffenden Eigenschaft geändert werden. So kann beispielsweise eine in einer Vorfahrtsklasse als **protected** deklarierte Eigenschaft im **public**- oder **published**-Abschnitt einer abgeleiteten Klasse neu deklariert werden. Eigenschaftsüberschreibungen können die Angaben **read**, **write**, **stored**, **default** und **nodefault** enthalten, durch die die entsprechende geerbte Direktive außer Kraft gesetzt wird. Mithilfe einer Überschreibung können Sie geerbte Zugriffsangaben ersetzen, fehlende Angaben hinzufügen oder den Gültigkeitsbereich einer Eigenschaft erweitern, jedoch keine Zugriffsbezeichner entfernen oder die Sichtbarkeit verringern. Optional kann auch mit der Direktive **implements** die Liste der implementierten Interfaces ergänzt werden, ohne die geerbten Interfaces zu entfernen.

Die folgenden Deklarationen zeigen, wie Eigenschaften überschrieben werden können:

```
type
```

```

TAncestor = class
  ...
  protected
    property Size: Integer read FSize;
    property Text: string read GetText write SetText;
    property Color: TColor read FColor write SetColor stored False;
  ...
end;

type

TDerived = class(TAncestor)
  ...
  protected
    property Size write SetSize;
  published
    property Text;
    property Color stored True default clBlue;
  ...
end;

```

Beim Überschreiben von `Size` wird die Angabe `write` hinzugefügt, damit der Wert der Eigenschaft geändert werden kann. Die Sichtbarkeit der Eigenschaften `Text` und `Color` wird von `protected` in `published` geändert. Für die Eigenschaft `Color` wird außerdem festgelegt, dass sie in der Formulardatei gespeichert wird, wenn sie einen anderen Wert als `clBlue` hat.

Wenn Sie beim Redeklarieren einer Eigenschaft einen Typbezeichner angeben, wird die geerbte Eigenschaft nicht überschrieben, sondern lediglich verdeckt. Dadurch wird eine neue Eigenschaft mit demselben Namen wie die geerbte erstellt. Diese Art der Deklaration muss immer vollständig vorgenommen werden. Stets muss zumindest eine Zugriffsangabe vorhanden sein.

Unabhängig davon, ob eine Eigenschaft in einer abgeleiteten Klasse verdeckt oder überschrieben wird, erfolgt die Suche nach der Eigenschaft immer statisch. Der deklarierte (also zur Compilierzeit bekannte) Typ der Variablen bestimmt die Interpretation des Eigenschaftsbezeichners. Aus diesem Grund wird nach dem Ausführen des folgenden Codes durch das Lesen oder Schreiben von `MyObject.Value` die Methode `Method1` bzw. `Method2` aufgerufen, obwohl `MyObject` eine Instanz von `TDescendant` enthält. Sie können aber durch eine Typumwandlung in `TDescendant` auf die Eigenschaften und Zugriffsangaben der abgeleiteten Klasse zugreifen.

```

type
  TAncestor = class
  ...
  property Value : Integer read Method1 write Method2;
end;

TDescendant = class(TAncestor)
  ...
  property Value : Integer read Method3 write Method4;
end;

var MyObject: TAncestor;
  ...
  MyObject := TDdescendant.Create;

```

Klasseneigenschaften

Auf Klasseneigenschaften kann ohne Objektreferenz zugegriffen werden. Methoden für den Zugriff auf Klasseneigenschaften müssen als klassenstatisch bzw. als Klassenfelder deklariert sein. Eine Klasseneigenschaft wird mit dem Schlüsselwörtern `class property` deklariert. Klasseneigenschaften können nicht als `published` deklariert werden und keine `stored`- oder `default`-Wertdefinitionen haben.

Mithilfe einer `class var`-Deklaration können Sie einen Block von statischen Klassenfeldern innerhalb einer Klassendeklaration erzeugen. Alle nach `class var` deklarierten Felder haben statische Speicherattribute. Ein `class var`-Block wird durch Folgendes abgeschlossen:

1. Eine weitere **class var**-Deklaration
2. Eine Prozedur- oder Funktionsdeklaration (Methodendeklaration) (siehe Seite 956) (einschließlich Klassenprozeduren und Klassenfunktionen)
3. Eine Eigenschaftsdeklaration (einschließlich Klasseneigenschaften)
4. Eine Konstruktor- oder Destruktor-Deklaration (siehe Seite 956)
5. Ein Sichtbarkeitsattribut (siehe Seite 948) (**public**, **private**, **protected**, **published**, **strict private** und **strict protected**)

Ein Beispiel:

```
type
  TMyClass = class
    strict private
      class var           // Felder müssen als Klassenfelder deklariert werden
        FRed: Integer;
        FGreen: Integer;
        FBlue: Integer;
      public              // Ende des class var-Blocks
        class property Red: Integer read FRed write FRed;
        class property Green: Integer read FGreen write FGreen;
        class property Blue: Integer read FBlue write FBlue;
    end;
```

Auf die obigen Klasseneigenschaften kann folgendermaßen zugegriffen werden:

```
TMyClass.Red := 0;
TMyClass.Blue := 0;
TMyClass.Green := 0;
```

Siehe auch

Klassen und Objekte (siehe Seite 948)

Felder (siehe Seite 954)

Methoden (siehe Seite 956)

Verschachtelte Typdeklarationen (siehe Seite 982)

Klassenreferenzen (siehe Seite 974)

Exceptions (siehe Seite 977)

Überladene Operatoren (siehe Seite 984)

Unterstützende Klassen (siehe Seite 987)

4.1.1.6.5 Ereignisse

Dieses Thema enthält Informationen zu folgenden Bereichen:

- Ereigniseigenschaften und Ereignisbehandlungsroutinen
- Mehrere Ereignisbehandlungsroutinen auslösen
- Multicast-Ereignisse (.NET)

Allgemeines zu Ereignissen

Ein Ereignis verknüpft ein Vorkommnis im System mit dem Quelltext, der auf dieses Vorkommnis antwortet. Das Vorkommnis löst die Ausführung einer Prozedur, Ereignisbehandlungsroutine genannt, aus. Die Ereignisbehandlungsroutine führt die Aufgaben aus, die als Antwort auf das Vorkommnis erforderlich sind. Ereignisse ermöglichen die Anpassung des Verhaltens einer Komponente während des Entwurfs oder zur Laufzeit. Zum Ändern des Verhaltens einer Komponente ersetzen Sie die Ereignisbehandlungsroutine durch eine eigene Ereignisbehandlungsroutine, die das gewünschte Verhalten enthält.

Ereigniseigenschaften und Ereignisbehandlungsroutinen

Komponenten, die in Delphi geschrieben sind, verwenden Eigenschaften zur Festlegung der Ereignisbehandlungsroutine, die ausgeführt werden soll, wenn das Ereignis eintritt. Per Konvention beginnt der Name einer Ereigniseigenschaft mit "On", und die Eigenschaft wird mit einem Feld anstelle von Lesen-/Schreibenmethoden implementiert. Der von der Eigenschaft gespeicherte Wert ist ein Methodenzeiger, der auf die Ereignisbehandlungsroutine zeigt.

Im folgenden Beispiel beinhaltet die Klasse `TObservedObject` ein `OnPing`-Ereignis des Typs `TPingEvent`. Im Feld `FOnPing` wird die Ereignisbehandlungsroutine gespeichert. Die Ereignisbehandlungsroutine `TListener.Ping` dieses Beispiels gibt 'TListener has been pinged!' aus.

```
4
Program EventDemo;
{$APPTYPE CONSOLE}

type
  TPingEvent = procedure of object;
  TObservedObject = class
  private
    FPing: TPingEvent;
  public
    property OnPing: TPingEvent read FPing write FPing;
  end;

  TListener = class
    procedure Ping;
  end;

procedure TListener.Ping;
begin
  writeln('TListener has been pinged.');
end;

var
  observedObject: TObservedObject;
  listener: TListener;

begin
  observedObject := TObservedObject.Create;
  listener := TListener.Create;

  observedObject.OnPing := listener.Ping;

  observedObject.OnPing; // soll 'TListener has been pinged.' ausgeben
  ReadLn; // vor dem Schließen Konsole auf Pause setzen
end.
```

Mehrere Ereignisbehandlungsroutinen auslösen

In Delphi für Win32 können Ereignisse nur einer einzigen Ereignisbehandlungsroutine zugeordnet werden. Wenn mehrere Ereignisbehandlungsroutinen als Reaktion auf ein Ereignis ausgeführt werden müssen, muss die dem Ereignis zugeordnete Ereignisbehandlungsroutine alle weiteren Ereignisbehandlungsroutinen aufrufen. Im folgenden Quelltext hat eine Unterklasse von `TListener`, `TListenerSubclass`, ihre eigene Ereignisbehandlungsroutine namens `Ping2`. In diesem Beispiel muss die Ereignisbehandlungsroutine `Ping2` die Ereignisbehandlungsroutine `TListener.Ping` explizit aufrufen, um sie als Reaktion auf das Ereignis `OnPing` auszulösen.

```
Program EventDemo2;
{$APPTYPE CONSOLE}

type
  TPingEvent = procedure of object;
  TObservedObject = class
  private
    FPing: TPingEvent;
```

```

public
  property OnPing: TPingEvent read FPing write FPing;
end;

TListener = class
  procedure Ping;
end;

TListenerSubclass = class (TListener)
  procedure Ping2;
end;

procedure TListener.Ping;
begin
  writeln('TListener has been pinged.');
end;

procedure TListenerSubclass.Ping2;
begin
  self.Ping;
  writeln('TListenerSubclass has been pinged.');
end;

var
  observedObject: TObservedObject;
  listener: TListenerSubclass;

begin
  observedObject := TObservedObject.Create;
  listener := TListenerSubclass.Create;

  observedObject.OnPing := listener.Ping2;

  observedObject.OnPing; // soll 'TListener has been pinged.' ausgeben
  // und dann 'TListenerSubclass has been pinged.'

  ReadLn; // vor dem Schließen Konsole auf Pause setzen
end.

```

Multicast-Ereignisse (nur .NET)

Auf der .NET-Plattform ermöglicht Delphi, dass für dasselbe Ereignis mehrere Ereignisbehandlungs Routinen angewendet werden. Ein Multicast-Ereignis ist ein Ereignis, das mehrere Ereignisbehandlungs Routinen auslösen kann. Durch Multicast-Ereignisse kann der Aufwand für die Wartung von komplexen Ereignissystemen reduziert und die Lesbarkeit und Flexibilität der Ereignisbehandlung verbessert werden.

Multicast-Ereignisse werden mit den Schlüsselwörtern **add** und **remove** deklariert, um das Feld oder die Methoden anzugeben, die für das Hinzufügen oder Entfernen von Behandlungs Routinen für ein Ereignis verwendet werden. Mit den Standardprozeduren **Include()** und **Exclude()** werden Ereignisbehandlungs Routinen zur Laufzeit ein- oder ausgeschlossen. Der folgende Quelltext zeigt die Deklaration einer Ereigniseigenschaft, die Multicast-Ereignisse verwendet.

```

Program NETEvents;
{$APPTYPE CONSOLE}

type
  TPingEvent = procedure of object;
  TObservedObject = class
  private
    FPing: TPingEvent;
  public
    property OnPing: TPingEvent add FPing remove FPing;
  end;

  TListener = class
    procedure Ping;
  end;

```

```

end;

TListenerSubclass = class (TListener)
  procedure Ping2;
end;

procedure TListener.Ping;
begin
  writeln('TListener has been pinged.');
end;

procedure TListenerSubclass.Ping2;
begin
  writeln('TListenerSubclass has been pinged.');
end;

var
  observedObject: TObservedObject;
  listener: TListener;
  listenerSubclass: TListenerSubclass;
  testEvent: TPingEvent;

begin
  observedObject := TObservedObject.Create;
  listener := TListener.Create;
  listenerSubclass := TListenerSubclass.Create;

  Include(observedObject.OnPing, listener.Ping);
  Include(observedObject.OnPing, listenerSubclass.Ping2);

  // testEvent := observedObject.OnPing;      // nicht zulässig
  observedObject.FPing(); // soll 'TListener has been pinged.' ausgeben

  ReadLn; // vor dem Schließen Konsole auf Pause setzen
end.

```

Die Eigenschaftsdeklaration `ObservedObject.OnPing` verwendet die Schlüsselwörter **add** und **remove** anstelle von **read** und **write**. Die Schlüsselwörter **add** und **remove** informieren den Compiler darüber, dass `OnClick` ein Multicast-Ereignis ist.

Eine Eigenschaft, die ein Multicast-Ereignis repräsentiert, kann nicht direkt gelesen oder geschrieben werden; auf sie kann nur mit den Standardprozeduren `Include()` und `Exclude()` zugegriffen werden.. Der Versuch eine Eigenschaft, die mit **add** und **remove** deklariert wurde, zu lesen, zu schreiben oder auszuführen, erzeugt beim Compilieren einen Fehler. Damit dieses Ereignis direkt ausgeführt werden kann, wird im Beispielquelltext das Feld `FPing` anstelle der Eigenschaft `OnPing` verwendet.

Siehe auch

[Klassen und Objekte](#) (siehe Seite 948)

[Eigenschaften](#) (siehe Seite 965)

[Methoden](#) (siehe Seite 956)

[Prozedurale Typen](#) (siehe Seite 914)

[Ereignisse erzeugen](#)

4.1.1.6 Klassenreferenzen

In manchen Situationen werden Operationen mit einer Klasse selbst und nicht mit ihren Instanzen (Objekten) durchgeführt. Dies geschieht beispielsweise, wenn Sie einen Konstruktor mit einer Klassenreferenz aufrufen. Sie können auf eine bestimmte Klasse immer über ihren Namen zugreifen. Manchmal müssen aber Variablen oder Parameter deklariert werden, die Klassen als Werte aufnehmen. Für diese Fälle benötigen Sie *Klassenreferenztypen*.

In diesem Thema werden folgende Bereiche erläutert:

- Klassenreferenztypen
- Klassen-Operatoren

Klassenreferenztypen

Klassenreferenztypen werden auch als Metaklassen bezeichnet. Die Definition erfolgt folgendermaßen:

```
class of Typ
```

Typ ist ein beliebiger Klassentyp. Der Bezeichner *Typ* gibt einen Wert des Typs `class of Typ` an. Ist *Typ1* ein Vorfahr von *Typ2*, dann ist `class of Typ2` zuweisungskompatibel zu `class of Typ1`. Die Anweisung:

```
type TClass = class of TObject;
var AnyObj: TClass;
```

deklariert eine Variable namens *AnyObj*, die eine beliebige Klassenreferenz aufnehmen kann. Ein Klassenreferenztyp darf nicht direkt in einer Variablen Deklaration oder Parameterliste definiert werden. Klassenreferenzvariablen kann auch der Wert `nil` zugewiesen werden.

Das folgende Beispiel (Konstruktor der Klasse `TCollection` in der Unit `Classes`) zeigt, wie Klassenreferenztypen verwendet werden:

```
type TCollectionItemClass = class of TCollectionItem;
  ...
constructor Create(ItemClass: TCollectionItemClass);
```

Diese Deklaration besagt, dass beim Erstellen eines `TCollection`-Instanzobjekts der Name einer von `TCollectionItem` abgeleiteten Klasse an den Konstruktor übergeben werden muss.

Klassenreferenztypen sind hilfreich, wenn Sie eine Klassenmethode oder einen virtuellen Konstruktor in einer Klasse oder einem Objekt aufrufen wollen, dessen aktueller Typ zur Compilierzeit nicht bekannt ist.

Konstruktoren und Klassenreferenzen

Ein Konstruktor kann mit einer Variablen eines Klassenreferenztyps aufgerufen werden. Auf diese Weise können Objekte erstellt werden, deren Typ zur Compilierzeit nicht bekannt ist. Zum Beispiel:

```
type TControlClass = class of TControl;

function CreateControl(ControlClass: TControlClass;
const ControlName: string; X, Y, W, H: Integer): TControl;
begin
  Result := ControlClass.Create(MainForm);
  with Result do
  begin
    Parent := MainForm;
    Name := ControlName;
    SetBounds(X, Y, W, H);
    Visible := True;
  end;
end;
```

Der Funktion `CreateControl` wird eine Klassenreferenz als Parameter übergeben. Er bestimmt, welche Art von Steuerelement erstellt wird. Der Parameter wird anschließend beim Aufruf des Konstruktors verwendet. Da Klassentypbezeichner Klassenreferenzwerte enthalten, kann im Aufruf von `CreateControl` der Bezeichner der Klasse angegeben werden, um eine Instanz vor ihr zu erstellen. Zum Beispiel:

```
CreateControl(TEdit, 'Edit1', 10, 10, 100, 20);
```

Konstruktoren, die mit Klassenreferenzen aufgerufen werden, sind normalerweise virtuell. Die entsprechende Implementierung wird anhand des beim Aufruf angegebenen Laufzeittyps aktiviert.

Klassen-Operatoren

Klassenmethoden (siehe Seite 956) arbeiten mit Klassenreferenzen. Jede Klasse erbt von TObject die zwei Klassenmethoden ClassType and ClassParent. Diese Methoden geben eine Referenz auf die Klasse eines Objekts und seiner direkten Vorfahrklasse zurück. Beide Methoden geben einen Wert des Typs TClass (TClass = class of TObject) zurück, der in einen spezielleren Typ umgewandelt werden kann. Alle Klassen erben außerdem die Methode InheritsFrom, mit der Sie ermitteln können, ob ein Objekt von einer bestimmten Klasse abgeleitet ist. Diese Methoden werden von den Operatoren **is** und **as** verwendet und normalerweise nicht direkt aufgerufen.

Der Operator is

Der Operator **is** führt eine dynamische Typprüfung durch. Mit ihm können Sie den aktuellen Laufzeittyp eines Objekts ermitteln. Der Ausdruck

Objekt is Klasse

gibt **True** zurück, wenn *Objekt* eine Instanz der angegebenen Klasse oder eines ihrer Nachkommen ist. Trifft dies nicht zu, wird **False** zurückgegeben (hat *Objekt* den Wert **nil**, ist der Rückgabewert ebenfalls **False**). Wenn der deklarierte Typ von *Objekt* nicht in Beziehung zu *Klasse* steht (wenn die Typen also unterschiedlich und nicht voneinander abgeleitet sind), gibt der Compiler eine Fehlermeldung aus. Zum Beispiel:

```
if ActiveControl is TEdit then TEdit(ActiveControl).SelectAll;
```

Diese Anweisung prüft zuerst, ob die Variable eine Instanz von **TEdit** oder einem ihrer Nachkommen ist, und führt anschließend eine Typumwandlung in **TEdit** durch.

Der Operator as

Der Operator **as** führt eine Typumwandlung mit Laufzeitprüfung durch. Der Ausdruck

Objekt as Klasse

gibt eine Referenz auf dasselbe Objekt wie *Objekt*, aber mit dem von *Klasse* angegebenen Typ zurück. Zur Laufzeit muss *Objekt* eine Instanz von *Klasse* oder einem ihrer Nachkommen bzw. **nil** sein. Andernfalls wird eine Exception ausgelöst. Wenn der deklarierte Typ von *Objekt* nicht in Beziehung zu *Klasse* steht (wenn die Typen also unterschiedlich und nicht voneinander abgeleitet sind), gibt der Compiler eine Fehlermeldung aus. Zum Beispiel:

```
with Sender as TButton do
begin
  Caption := '&Ok';
  OnClick := OkClick;
end;
```

Die Regeln der Auswertungsreihenfolge machen es häufig erforderlich, **as**-Typumwandlungen in Klammern zu setzen. Zum Beispiel:

```
(Sender as TButton).Caption := '&Ok';
```

Siehe auch

Klassen und Objekte (siehe Seite 948)

Felder (siehe Seite 954)

Methoden (siehe Seite 956)

Eigenschaften (siehe Seite 965)

Verschachtelte Typdeklarationen (siehe Seite 982)

Exceptions (siehe Seite 977)

Überladene Operatoren (siehe Seite 984)

Unterstützende Klassen (siehe Seite 987)

4.1.1.6.7 Exceptions

In diesem Thema werden folgende Bereiche erläutert:

- Überblick über die Konzepte von Exceptions und der Exception-Behandlung
- Exception-Typen deklarieren
- Exceptions auslösen und behandeln

Allgemeines zu Exceptions

Eine Exception wird ausgelöst, wenn die normale Programmausführung durch einen Fehler oder ein anderes Ereignis unterbrochen wird. Die Steuerung wird dadurch an eine Exception-Behandlungsroutine übergeben. Mit ihrer Hilfe kann die normale Programmlogik von der Fehlerbehandlung getrennt werden. Da Exceptions Objekte sind, können sie durch Vererbung in einer Hierarchie organisiert werden. Sie bringen bestimmte Informationen (z. B. eine Fehlermeldung) von der Stelle im Programm, an der sie ausgelöst wurden, zu dem Punkt, an dem sie behandelt werden.

Wenn die Unit `SysUtils` in einer Anwendung verwendet wird, werden die meisten Laufzeitfehler automatisch in Exceptions konvertiert. Viele Fehler, die andernfalls zum Beenden der Anwendung führen (z. B. Speichermangel, Division durch Null oder allgemeine Schutzverletzungen), können so abgefangen und behandelt werden.

Einsatzmöglichkeiten für Exceptions

Exceptions ermöglichen das Abfangen von Laufzeitfehlern, die sonst einen Programmabbruch zur Folge hätten oder umständliche Bedingungsanweisungen erfordern würden. Die Semantik der Exception-Behandlung stellt bestimmte Anforderungen, die zu Abzügen in den Bereichen Code-/Datengröße und Laufzeitleistung führen. Zwar kann für fast alle Probleme oder Fehler eine Exception generiert und praktisch jeder Quelltextblock durch eine umgebende `try...except`- oder `try...finally`-Anweisung geschützt werden, in der Praxis sollte dieses Vorgehen aber auf Sonderfälle beschränkt bleiben.

Die Exception-Behandlung eignet sich für Fehler, die selten auftreten oder sich nur schwer eingrenzen lassen, die aber schwerwiegende Folgen haben können (z. B. einen Programmabsturz). Sie kann auch für Fehlerbedingungen eingesetzt werden, die sich nur mit großem Aufwand in `if...then`-Anweisungen testen lassen. Außerdem bietet sie sich für Exceptions an, die vom Betriebssystem oder von Routinen ausgelöst werden, auf deren Quellcode kein Zugriff möglich ist.

Normalerweise kommen Exceptions bei Hardware-, Speicher-, E/A- und Betriebssystemfehlern zum Einsatz. Bedingungsanweisungen sind oft die beste Methode für einen Fehlertest. Ein Beispiel:

```
try
  AssignFile(F, FileName);
  Reset(F);      // Löst die Exception EInOutError aus, wenn die Datei nicht gefunden wird
except
  on Exception do ...
end;
```

Mit der folgenden Anweisung können Sie den Aufwand der Exception-Behandlung vermeiden:

```
if FileExists(FileName) then      // Gibt False zurück, wenn die Datei nicht gefunden wird; löst
  keine Exception aus
begin
  AssignFile(F, FileName);
  Reset(F);
end;
```

Assertions stellen eine weitere Möglichkeit dar, eine Boolesche Bedingung im Quelltext zu testen. Wenn eine `Assert`-Anweisung fehlschlägt, wird entweder das Programm mit einem Laufzeitfehler angehalten oder (bei Verwendung der Unit `SysUtils`) eine `EAssertionFailed`-Exception ausgelöst. Assertions sollten nur zum Testen von Bedingungen verwendet werden, deren Auftreten unwahrscheinlich ist.

Exception-Typen deklarieren

Exception-Typen werden genau wie andere Klassen deklariert. Eigentlich können Sie eine Instanz einer beliebigen Klasse als Exception verwenden. Es ist aber zu empfehlen, Exceptions immer von der Klasse `Exception` (Unit `SysUtils`) abzuleiten.

Mithilfe der Vererbung können Exceptions in Familien organisiert werden. Die folgenden Deklarationen in `SysUtils` definieren beispielsweise eine Familie von Exception-Typen für mathematische Fehler:

```
type
  EMathError = class(Exception);
  EInvalidOp = class(EMathError);
  EZeroDivide = class(EMathError);
  EOverflow = class(EMathError);
  EUnderflow = class(EMathError);
```

Aufgrund dieser Deklarationen können Sie eine Behandlungsmethode für `EMathError` bereitstellen und in dieser auch `EInvalidOp`, `EZeroDivide`, `EOverflow` und `EUnderflow` behandeln.

In Exception-Klassen sind manchmal auch Felder, Methoden oder Eigenschaften definiert, die zusätzliche Informationen über den Fehler liefern. Zum Beispiel:

```
type EInOutError = class(Exception)
  ErrorCode: Integer;
end;
```

Exceptions auslösen und behandeln

Um ein Exception-Objekt auszulösen, verwenden Sie eine Instanz der Exception-Klasse mit einer `raise`-Anweisung. Zum Beispiel:

```
raise EMathError.Create;
```

Im Allgemeinen hat eine `raise`-Anweisung folgende Form:

raise Objekt at Adresse

`Objekt` und `at Adresse` sind optional. Bei Angabe einer Adresse kann es sich um einen beliebigen Ausdruck handeln, der einen Zeigertyp ergibt. In der Regel handelt es sich um einen Zeiger auf eine Prozedur oder eine Funktion. Zum Beispiel:

```
raise Exception.Create('Fehlender Parameter') at @MyFunction;
```

Mithilfe dieser Option kann die Exception an einem früheren Punkt im Stack ausgelöst werden.

Wenn eine Exception ausgelöst (d. h. in einer `raise`-Anweisung angegeben) wird, unterliegt sie einer speziellen Behandlungslogik. Die Programmsteuerung wird durch eine `raise`-Anweisung nicht auf normale Weise zurückgegeben. Sie wird stattdessen an die innerste Behandlungsroutine übergeben, die Exceptions der jeweiligen Klasse verarbeiten kann. Bei dieser handelt es sich um die Routine, deren `try...except`-Block zuletzt ausgeführt, aber noch nicht beendet wurde.

In der folgenden Funktion wird ein String in einen Integer-Wert konvertiert. Wenn dieser Wert nicht innerhalb eines bestimmten Bereichs liegt, wird eine `ERangeError`-Exception ausgelöst.

```
function StrToIntRange(const S: string; Min, Max: Longint): Longint;
begin
  Result := StrToInt(S); // StrToInt ist in SysUtils deklariert
  if (Result < Min) or (Result > Max) then
    raise ERangeError.CreateFmt('%d is not within the valid range of %d..%d', [Result, Min, Max]);
end;
```

Beachten Sie die Methode `CreateFmt`, die in der `raise`-Anweisung aufgerufen wird. Die Klasse `Exception` und ihre Nachkommen verfügen über spezielle Konstruktoren, um Fehlermeldungen und Kontext-IDs zu erstellen.

Eine ausgelöste Exception wird nach ihrer Behandlung automatisch wieder freigegeben. Versuchen Sie daher niemals, Exceptions manuell freizugeben.

Anmerkung: Das Auslösen einer Exception im **initialization**-Abschnitt einer Unit führt nicht zum gewünschten Ergebnis. Die normale Exception-Unterstützung wird durch die Unit `SysUtils` eingebunden, die daher zuerst initialisiert werden muss. Wenn während der Initialisierung eine Exception ausgelöst wird, werden alle initialisierten Units (einschließlich `SysUtils`) finalisiert, und die Exception wird erneut ausgelöst. Anschließend wird sie abgefangen und behandelt. Dabei wird das Programm normalerweise unterbrochen. Ähnlich führt das Auslösen einer Exception im **finalization**-Abschnitt einer Unit eventuell nicht zum gewünschten Ergebnis, falls bei Auslösen der Exception `SysUtils` bereits abgeschlossen wurde.

Die Anweisung `try...except`

Exceptions werden mithilfe von `try...except`-Anweisungen behandelt. Zum Beispiel:

```
try
  X := Y/Z;
  except
    on EZeroDivide do HandleZeroDivide;
  end;
```

Zuerst wird im `try`-Block die Division `Y / Z` durchgeführt. Tritt dabei eine `EZeroDivide`-Exception (Division durch Null) auf, wird die Behandlungsroutine `HandleZeroDivide` aufgerufen.

Die Syntax einer `try...except`-Anweisung lautet folgendermaßen:

try Anweisungsliste except ExceptionBlock end

Anweisungsliste ist eine Folge beliebiger Anweisungen, die durch einen Strichpunkt voneinander getrennt sind. **ExceptionBlock** ist entweder

- eine weitere Anweisungsfolge oder
- eine Folge von Exception-Behandlungsroutinen, optional mit nachfolgendem

else Anweisungsliste

Eine Exception-Behandlungsroutine hat folgende Form:

on Bezeichner: Typ do Anweisung

Bezeichner ist optional und kann ein beliebiger Bezeichner sein. **Typ** ist ein für die Exception verwendeter Typ, und **Anweisung** ist eine beliebige Anweisung.

In einer `try...except`-Anweisung werden zuerst die Programmzeilen in **Anweisungsliste** ausgeführt. Werden dabei keine Exceptions ausgelöst, wird **ExceptionBlock** ignoriert und die Steuerung an den nächsten Programmteil übergeben.

Tritt bei der Ausführung der Anweisungsliste eine Exception auf (entweder durch eine `raise`-Anweisung oder eine aufgerufene Prozedur bzw. Funktion), versucht das Programm, diese zu behandeln:

- Stimmt eine der Behandlungsroutinen im Exception-Block mit der betreffenden Exception überein, wird die Steuerung an diese Routine übergeben. Eine Übereinstimmung liegt vor, wenn der Typ in der Behandlungsroutine der Klasse der Exception oder eines ihrer Nachkommen entspricht.
- Wenn keine Behandlungsroutine existiert, wird die Steuerung an die Anweisung in der **else**-Klausel übergeben (falls vorhanden).
- Besteht der Exception-Block lediglich aus einer Folge von Anweisungen (ohne Exception-Behandlungsroutinen), wird die Steuerung an die erste Anweisung in der Liste übergeben.

Trifft keine dieser Bedingungen zu, wird die Suche im Exception-Block der zuletzt ausgeführten und noch nicht beendeten `try...except`-Anweisung fortgesetzt. Kann dort keine entsprechende Behandlungsroutine, **else**-Klausel oder Anweisungsliste gefunden werden, wird die nächste `try...except`-Anweisung durchsucht usw. Ist die Exception bei Erreichen des äußersten `try...except`-Blocks immer noch nicht behandelt worden, wird das Programm beendet.

Beim Behandeln einer Exception wird der Aufruf-Stack nach oben bis zu der Prozedur oder Funktion durchlaufen, in der sich die `try...except`-Anweisung befindet, in der die Behandlung durchgeführt wird. Die Steuerung wird dann an die entsprechende Exception-Behandlungsroutine, `else`-Klausel oder Anweisungsliste übergeben. Bei diesem Vorgang werden alle Prozedur- und Funktionsaufrufe verworfen, die nach dem Eintritt in den `try...except`-Block stattgefunden haben. Anschließend wird das Exception-Objekt durch einen Aufruf seines Destruktors `Destroy` automatisch freigegeben, und die Programmausführung wird mit der nächsten Anweisung nach dem `try...except`-Block fortgesetzt. Das Objekt wird auch automatisch freigegeben, wenn die Behandlungsroutine durch einen Aufruf der Standardprozedur `Exit`, `Break` oder `Continue` verlassen wird.

Im folgenden Beispiel sind drei Behandlungsroutinen definiert. Die erste behandelt Divisionen durch Null, die zweite Überläufe, und die dritte alle anderen mathematischen Exceptions. Der Typ `EMathError` ist zuletzt aufgeführt, da er der Vorfahr der anderen beiden Exception-Klassen ist. Würde er an erster Stelle genannt, käme es nie zu einem Aufruf der beiden anderen Routinen.

```
try
  ...
except
  on EZeroDivide do HandleZeroDivide;
  on EOverflow do HandleOverflow;
  on EMathError do HandleMathError;
end;
```

Vor dem Namen der Exception-Klasse kann optional ein Bezeichner angegeben werden. Dieser Bezeichner dient in der auf `on...do` folgenden Anweisung zum Zugriff auf das Exception-Objekt. Der Gültigkeitsbereich des Bezeichners ist auf diese Anweisung beschränkt. Zum Beispiel:

```
try
  ...
except
  on E: Exception do ErrorDialog(E.Message, E.HelpContext);
end;
```

Im Exception-Block kann auch eine `else`-Klausel angegeben werden. Dort werden alle Exceptions behandelt, die nicht von den Behandlungsroutinen für den Block abgedeckt werden. Zum Beispiel:

```
try
  ...
except
  on EZeroDivide do HandleZeroDivide;
  on EOverflow do HandleOverflow;
  on EMathError do HandleMathError;
else
  HandleAllOthers;
end;
```

In diesem Fall werden in der `else`-Klausel alle Exceptions außer `EMathError` behandelt.

Ein Exception-Block, der nur eine Liste von Anweisungen, jedoch keine Behandlungsroutinen enthält, behandelt alle Exceptions. Zum Beispiel:

```
try
  ...
except
  HandleException;
end;
```

Hier behandelt die Routine `HandleException` alle Exceptions, die bei der Ausführung der Anweisungen zwischen `try` und `except` ausgelöst werden.

Exceptions erneut auslösen

Wenn Sie das reservierte Wort `raise` ohne nachfolgende Objektreferenz in einem Exception-Block angeben, wird die aktuell behandelte Exception nochmals ausgelöst. Auf diese Weise kann in einer Behandlungsroutine begrenzt auf einen Fehler reagiert und anschließend die Exception erneut ausgelöst werden. Diese Möglichkeit ist hilfreich, wenn in einer Prozedur oder Funktion nach Auftreten einer Exception Aufräumarbeiten durchgeführt werden sollen (z. B. Objekte oder Ressourcen freigeben).

Im folgenden Beispiel wird von der Funktion `GetFileList` ein `TStringList`-Objekt erstellt und mit den Namen der Dateien im übergebenen Pfad gefüllt:

```
function GetFileList(const Path: string): TStringList;
var
  I: Integer;
  SearchRec: TSearchRec;
begin
  Result := TStringList.Create;
  try
    I := FindFirst(Path, 0, SearchRec);
    while I = 0 do
      begin
        Result.Add(SearchRec.Name);
        I := FindNext(SearchRec);
      end;
  except
    Result.Free;
    raise;
  end;
end;
```

In dieser Funktion wird ein `TStringList`-Objekt erstellt und mithilfe der Funktionen `FindFirst` und `FindNext` (Unit `SysUtils`) mit Werten gefüllt. Tritt dabei ein Fehler auf (z. B. aufgrund eines ungültigen Pfades oder wegen Speichermangels), muss das neue Objekt freigegeben werden, da es der aufrufenden Routine noch nicht bekannt ist. Aus diesem Grund muss die Initialisierung der String-Liste in einer `try...except`-Anweisung durchgeführt werden. Bei einer Exception wird das Objekt im Exception-Block freigegeben und anschließend die Exception erneut ausgelöst.

Verschachtelte Exceptions

In einer Exception-Behandlungsroutine können wiederum Exceptions ausgelöst und behandelt werden. Solange dieser Vorgang ebenfalls innerhalb der Routine stattfindet, hat er keinen Einfluss auf die ursprüngliche Exception. Wenn die zweite Exception jedoch die Routine verlässt, geht die Original-Exception verloren. Ein Beispiel:

```
type
  ETrigError = class(EMathError);
  function Tan(X: Extended): Extended;
begin
  try
    Result := Sin(X) / Cos(X);
  except
    on EMathError do
      raise ETrigError.Create('Ungültiges Argument für Tan');
  end;
end;
```

Wenn während der Ausführung von `Tan` eine `EMathError`-Exception auftritt, wird in der Behandlungsroutine eine `ETrigError`-Exception ausgelöst. Da in `Tan` keine Routine für `ETrigError` definiert ist, verlässt die Exception die Behandlungsroutine, und die ursprüngliche `EMathError`-Exception wird freigegeben. Für die aufrufende Routine stellt sich der Vorgang so dar, als ob die Funktion `Tan` eine `ETrigError`-Exception ausgelöst hat.

Die Anweisung `try...finally`

In manchen Situationen muss sichergestellt sein, dass bestimmte Operationen auch bei Auftreten einer Exception vollständig abgeschlossen werden. Wenn beispielsweise in einer Routine eine Ressource zugewiesen wird, ist es sehr wichtig, dass sie unabhängig von der Beendigung der Routine wieder freigegeben wird. In diesen Fällen können `try...finally`-Anweisungen verwendet werden.

Das folgende Beispiel zeigt, wie eine Datei auch dann wieder geschlossen werden kann, wenn beim Öffnen oder Bearbeiten eine Exception auftritt:

```
Reset(F);
try
```

```
... // Datei F verarbeiten
finally
  CloseFile(F);
end;
```

Eine `try...finally`-Anweisung hat folgende Syntax:

try Anweisungsliste1 finally Anweisungsliste2 end

Jede `Anweisungsliste` setzt sich aus einer Folge von Anweisungen zusammen, die durch einen Strichpunkt voneinander getrennt sind. In einem `try...finally`-Block werden zuerst die Programmzeilen in `Anweisungsliste1` (**try**-Klausel) ausgeführt. Wenn dabei keine Exceptions auftreten, wird anschließend `Anweisungsliste2` (**finally**-Klausel) ausgeführt. Bei einer Exception wird die Steuerung an `Anweisungsliste2` übergeben und danach die Exception erneut ausgelöst. Befindet sich ein Aufruf der Standardprozedur `Exit`, `Break` oder `Continue` in `Anweisungsliste1`, wird dadurch automatisch `Anweisungsliste2` aufgerufen. Daher wird die **finally**-Klausel unabhängig davon, wie der **try**-Block beendet wird, immer ausgeführt.

Wenn eine Exception ausgelöst, aber in der **finally**-Klausel nicht behandelt wird, führt sie aus der `try...finally`-Anweisung hinaus, und jede zuvor in der **try**-Klausel ausgelöste Exception geht verloren. In der **finally**-Klausel sollten daher alle lokal ausgelösten Exceptions behandelt werden, damit die Behandlung anderer Exceptions nicht gestört wird.

Exception-Standardklassen und -Standardroutinen

In den Units `SysUtils` und `System` sind verschiedene Standardroutinen für die Exception-Behandlung deklariert (z. B. `ExceptObject`, `ExceptAddr` und `ShowException`). `SysUtils`, `System` und andere Units enthalten auch zahlreiche Exception-Klassen (außer `OutlineError`), die von `Exception` abgeleitet sind.

Die Klasse `Exception` verfügt über die Eigenschaften `Message` und `HelpContext`, durch die eine Fehlerbeschreibung und eine Kontext-ID für die kontextbezogene Online-Dokumentation übergeben werden kann. Außerdem definiert sie verschiedene Konstruktor-Methoden, mit denen Fehlerbeschreibungen und Kontext-IDs auf unterschiedliche Arten angegeben werden können.

Siehe auch

Klassen und Objekte ([siehe Seite 948](#))

Felder ([siehe Seite 954](#))

Methoden ([siehe Seite 956](#))

Eigenschaften ([siehe Seite 965](#))

Verschachtelte Typdeklarationen ([siehe Seite 982](#))

Klassenreferenzen ([siehe Seite 974](#))

Überladene Operatoren ([siehe Seite 984](#))

Unterstützende Klassen ([siehe Seite 987](#))

4.1.1.6.8 Verschachtelte Typdeklarationen

Typdeklarationen können in Klassendeklarationen verschachtelt werden. Verschachtelte Typen sind sowohl im .NET Framework als auch bei der objektorientierten Programmierung gebräuchlich. Mit ihrer Hilfe lassen sich Typen, die eng miteinander verwandt sind, in Gruppen zusammenfassen und Namenskollisionen vermeiden. Diese Syntax zur Deklaration von verschachtelten Typen kann auch für den Win32-Delphi-Compiler verwendet werden.

Verschachtelte Typen deklarieren

`VerschachtelteTypdeklaration` entspricht der unter Typdeklaration ([siehe Seite 923](#)) beschriebenen Syntax zum Deklarieren von Typen.

```
type
  Klassenname = class [abstract | sealed] (Vorfahrtyp)
```

Elementliste

```
type
  VerschachtelteTypdeklaration
```

```
  Elementliste
end;
```

Der Deklarationsabschnitt für einen verschachtelten Typ wird durch das nächste Token, das kein Bezeichner ist (z. B. **procedure**, **class** oder **type**), oder durch ein beliebiges Sichtbarkeitsattribut abgeschlossen.

Für verschachtelte Typen und ihre Containertypen gelten die normalen Zugriffsregeln. Ein verschachtelter Typ kann auf eine Instanzvariable (Feld, Eigenschaft oder Methode) seiner Containerklasse zugreifen, benötigt dazu allerdings eine Objektreferenz. Der Zugriff auf Klassenfelder, Klasseneigenschaften und klassenstatische Methoden ist ohne Objektreferenz möglich, es gelten aber die normalen Sichtbarkeitsregeln von Delphi.

Die Größe der Containerklasse erhöht sich durch verschachtelte Typen nicht. Bei der Erstellung einer Instanz der Containerklasse wird keine Instanz eines verschachtelten Typs erzeugt. Verschachtelte Typen sind mit ihren Containerklassen nur über den Kontext der Deklaration verknüpft.

Verschachtelte Klassen deklarieren und verwenden

Das folgende Beispiel veranschaulicht, wie man Felder und Methoden einer verschachtelten Klasse deklariert und auf sie zugreift.

```
type
  TOuterClass = class
    strict private
      myField: Integer;

    public
      type
        TInnerClass = class
          public
            myInnerField: Integer;
            procedure innerProc;
          end;

          procedure outerProc;
        end;
```

Um die Methode `innerProc` der inneren Klasse zu implementieren, muss ihr Name mit dem Namen der äußeren Klasse qualifiziert werden. Beispiel:

```
procedure TOuterClass.TInnerClass.innerProc;
begin
  ...
end;
```

Der Zugriff auf Elemente des verschachtelten Typs kann wie bei normalen Klassenelementen mittels Punktnotation erfolgen. Beispiel:

```
var
  x: TOuterClass;
  y: TOuterClass.TInnerClass;

begin
  x := TOuterClass.Create;
  x.outerProc;
  ...
  y := TOuterClass.TInnerClass.Create;
  y.innerProc;
```

Verschachtelte Konstanten

Konstanten können in Klassentypen auf die gleiche Weise wie verschachtelte Typen deklariert werden. Ein

Deklarationsabschnitt für eine verschachtelte Konstante wird wie bei einem verschachtelten Typ durch ein reserviertes Wort oder ein Sichtbarkeitsattribut abgeschlossen. Typisierte Konstanten werden nicht unterstützt. Es können deshalb keine verschachtelten Konstanten eines Werttyps (wie Currency oder TDateTime) deklariert werden.

Verschachtelte Konstanten können einen beliebigen einfachen Typ haben (ordinal, ordinaler Teilbereich, Aufzählung, String und reell).

Das folgende Beispiel veranschaulicht die Deklaration von verschachtelten Konstanten:

```
4
type
  TMyClass = class
  const
    x = 12;
    y = TMyClass.x + 23;
  procedure Hello;
  private
    const
      s = 'Eine String-Konstante';
  end;

begin
  writeln(TMyClass.y); // Schreibt den Wert von y, 35.
end.
```

Siehe auch

[Klassen und Objekte](#) (siehe Seite 948)

[Felder](#) (siehe Seite 954)

[Methoden](#) (siehe Seite 956)

[Eigenschaften](#) (siehe Seite 965)

[Klassenreferenzen](#) (siehe Seite 974)

[Exceptions](#) (siehe Seite 977)

[Überladene Operatoren](#) (siehe Seite 984)

[Unterstützende Klassen](#) (siehe Seite 987)

4.1.1.6.9 Überladene Operatoren

Dieses Thema enthält einen Überblick über die Operatormethoden von Delphi und beschreibt, wie Operatoren überladen werden.

Allgemeines zum Überladen von Operatoren

Delphi für .NET und Delphi für Win32 ermöglicht es, bestimmte Funktionen, auch "Operatoren" genannt, innerhalb von Record-Deklarationen zu überladen. Delphi für .NET lässt außerdem das Überladen in Klassendeklarationen zu. Der Name der Operatorfunktion wird dabei einem Symbol im Quelltext zugeordnet. Beispielsweise entspricht das Symbol **+** dem Operator **Add**. Der Compiler vergleicht den jeweiligen Kontext (Rückgabetyp und Typ der im Aufruf verwendeten Parameter) mit der Signatur der Operatorfunktion, und ruft den entsprechenden überladenen Operator auf. Die folgende Tabelle enthält die Delphi-Operatoren, die überladen werden können.

Operator	Kategorie	Deklarationssignatur	Symbol
Implicit	Konvertierung	Implicit(a : Typ): Ergebnistyp;	Implizite Typumwandlung
Explicit	Konvertierung	Explicit(a: Typ): Ergebnistyp;	Explizite Typumwandlung
Negative	Unär	Negative(a: Typ): Ergebnistyp;	-
Positive	Unär	Positive(a: Typ): Ergebnistyp;	+

Inc	Unär	Inc(a: Typ): Ergebnistyp;	Inc
Dec	Unär	Dec(a: Typ): Ergebnistyp	Dec
LogicalNot	Unär	LogicalNot(a: Typ): Ergebnistyp;	not
BitwiseNot	Unär	BitwiseNot(a: Typ): Ergebnistyp;	not
Trunc	Unär	Trunc(a: Typ): Ergebnistyp;	Trunc
Round	Unär	Round(a: Typ): Ergebnistyp;	Round
Equal	Vergleich	Equal(a: Typ; b: Typ): Boolean;	=
NotEqual	Vergleich	NotEqual(a: Typ; b: Typ): Boolean;	<>
GreaterThan	Vergleich	GreaterThan(a: Typ; b: type) Boolean;	>
GreaterThanOrEqual	Vergleich	GreaterThanOrEqual(a: Typ; b: Typ): Ergebnistyp;	>=
LessThan	Vergleich	LessThan(a: Typ; b: Typ): Ergebnistyp;	<
LessThanOrEqual	Vergleich	LessThanOrEqual(a: Typ; b: Typ): Ergebnistyp;	<=
Add	Binär	Add(a: Typ; b: Typ): Ergebnistyp;	+
Subtract	Binär	Subtract(a: Typ; b: Typ): Ergebnistyp;	-
Multiply	Binär	Multiply(a: Typ; b: Typ): Ergebnistyp;	*
Divide	Binär	Divide(a: Typ; b: Typ): Ergebnistyp;	/
IntDivide	Binär	IntDivide(a: Typ; b: Typ): Ergebnistyp;	div
Modulus	Binär	Modulus(a: Typ; b: Typ): Ergebnistyp;	mod
LeftShift	Binär	LeftShift(a: Typ; b: Typ): Ergebnistyp;	shl
RightShift	Binär	RightShift(a: Typ; b: Typ): Ergebnistyp;	shr
LogicalAnd	Binär	LogicalAnd(a: Ergebnistyp; b: Typ):	and
LogicalOr	Binär	LogicalOr(a: Ergebnistyp; b: Typ):	or
LogicalXor	Binär	LogicalXor(a: Ergebnistyp; b: Typ):	xor
BitwiseAnd	Binär	BitwiseAnd(a: Ergebnistyp; b: Typ):	and
BitwiseOr	Binär	BitwiseOr(a: Ergebnistyp; b: Typ):	or
BitwiseXor	Binär	BitwiseXor(a: Ergebnistyp; b: Typ):	xor

Nur die in der Tabelle aufgeführten Operatoren können für eine Klasse oder einen Record definiert werden.

Es ist nicht möglich, eine überladene Operatormethode im Quelltext über ihren Namen zu referenzieren. Um auf eine bestimmte Operatormethode einer bestimmten Klasse oder eines Records zuzugreifen, müssen explizite Typumwandlungen für alle Operanden verwendet werden. Operatorbezeichner werden nicht in die Elementliste der Klasse bzw. des Record aufgenommen.

Es bestehen keine Anforderungen bezüglich der distributiven oder kommutativen Eigenschaften der Operation. Bei binären Operatoren ist der erste Parameter immer der linke Operand und der zweite Parameter der rechte Operand. Sind keine

Klammern vorhanden, erfolgt die Auswertung von links nach rechts.

Die Zuordnung von Operatormethoden erfolgt mittels der verfügbaren Operatoren der in der Operation verwendeten Typen (dies gilt auch für geerbte Operatoren). Wenn eine Operation mit den zwei unterschiedlichen Typen A und B für Typ A eine implizite Umwandlung in Typ B (und umgekehrt) vorsieht, ist keine Eindeutigkeit gegeben. Implizite Umwandlungen sollten nur wenn unbedingt nötig vorgenommen werden. Reflexivität ist zu vermeiden. Im vorliegenden Fall ist es besser, ohne Kenntnis von Typ A eine implizite Umwandlung von Typ B in Typ A zu veranlassen (oder umgekehrt).

Grundsätzlich dürfen Operatoren ihre Operanden nicht verändern. Stattdessen muss ein neuer Wert zurückgegeben werden, der sich aus der Durchführung der Operation für die Parameter ergibt.

Überladene Operatoren werden meist in Records (Werttypen) verwendet. Im Gegensatz zu den meisten Werttypen haben nur wenige Klassen im .NET Framework überladene Operatoren.

Überladene Operatoren deklarieren

Überladene Operatoren werden in Klassen oder Records mit der folgenden Syntax deklariert:

```
type
  Typename = [class | record]
    class operator conversionOp(a: Typ): Ergebnistyp;
    class operator unärerOp(a: Typ): Ergebnistyp;
    class operator Vergleichsop(a: Typ; b: Typ): Boolean;
    class operator binärerOp(a: Typ; b: Typ): Ergebnistyp;
  end;
```

In der Implementierung überladener Operatoren muss ebenfalls die **class operator**-Syntax verwendet werden:

```
class operator Typname.UmwandlungOp(a: Typ): Ergebnistyp;
class operator Typname.unärerOp(a: Typ): Ergebnistyp;
class operator Typname.Vergleichsop(a: Typ; b: Typ): Boolean;
class operator Typname.binärerOp(a: Typ; b: Typ): Ergebnistyp;
```

Nachstehend finden Sie einige Beispiele für überladene Operatoren.

```
type
  TMyClass = class
    class operator Add(a, b: TMyClass): TMyClass;           // zweier Operanden des Typs TMyClass
    class operator Subtract(a, b: TMyClass): TMyClass;      // Subtraktion des Typs TMyClass
    class operator Implicit(a: Integer): TMyClass;          // Implizite Umwandlung eines Integers
  in den Typ TMyClass
    class operator Implicit(a: TMyClass): Integer;          // Implizite Umwandlung von TMyClass
  in einem Inter
    class operator Explicit(a: Double): TMyClass;          // Explizit Umwandlung eines
  Double-Werts in TMyClass
  end;

  // Beispielimplementierung für Add
  class operator TMyClass.Add(a, b: TMyClass): TMyClass;
  begin
    // ...
  end;

  var
  x, y: TMyClass;
  begin
    x := 12;          // Implizite Umwandlung aus Integer
    y := x + x;      // Aufruf von TMyClass.Add(a, b: TMyClass): TMyClass
    b := b + 100;    // Aufruf von TMyClass.Add(b, TMyClass.Implicit(100))
  end;
```

Siehe auch

Klassen und Objekte (siehe Seite 948)

Felder (siehe Seite 954)

Methoden (siehe Seite 956)

Eigenschaften (siehe Seite 965)

Verschachtelte Typdeklarationen (siehe Seite 982)

Unterstützende Klassen (siehe Seite 987)

Klassenreferenzen (siehe Seite 974)

Exceptions (siehe Seite 977)

4.1.1.6.10 Unterstützende Klassen

Dieses Thema befasst sich mit der Syntax für die Deklaration von unterstützenden Klassen.

Allgemeines zu unterstützenden Klassen

Eine unterstützende Klasse ist ein Typ, der für eine andere Klasse zusätzliche Methodennamen und Eigenschaften bereitstellt. Diese können im Kontext der verknüpften Klasse (und deren Nachkommen) verwendet werden. Unterstützende Klassen ermöglichen es, eine Klasse ohne Vererbung zu erweitern. Sie vergrößern den Gültigkeitsbereich, den der Compiler bei der Zuordnung von Bezeichnern verwendet. Bei der Deklaration einer unterstützenden Klasse geben Sie deren Namen sowie den Namen der Klasse an, die erweitert werden soll. Eine unterstützende Klasse kann überall dort eingesetzt werden, wo die Verwendung der mit ihr verknüpften Klasse zulässig ist. Der Gültigkeitsbereich, den der Compiler für die Zuordnung verwendet, umfasst dann die ursprüngliche Klasse und die unterstützende Klasse.

Unterstützende Klassen erweitern eine Klasse, sollten aber nicht als Entwurfs-Tool für die Erstellung neuen Quelltextes eingesetzt werden. Ihr ausschließlicher Einsatzzweck ist die sprach- und plattformspezifische Laufzeitbindung.

Deklarationssyntax

Die Syntax für die Deklaration einer unterstützenden Klasse lautet:

```
type
  Bezeichnername = class helper [(Vorfahrenliste)] for Klassentyp Bezeichnername
    Elementliste
  end;
```

Die *Vorfahrenliste* ist optional.

Eine unterstützende Klasse kann keine Instanzdaten deklarieren. Klassenfelder (siehe Seite 954) sind aber zulässig.

Hinsichtlich der Sichtbarkeit und der Syntax für die *Elementliste* gelten dieselben Regeln wie für normale Klassentypen.

In einem Klassentyp können mehrere unterstützende Klassen definiert und verknüpft werden. An einer Stelle im Quelltext kann aber jeweils nur eine unterstützende Klasse angegeben werden. Es wird die Klasse verwendet, die im Gültigkeitsbereich zuerst gefunden wird. Der Gültigkeitsbereich für unterstützende Klassen wird wie in Delphi üblich bestimmt (die **uses**-Klausel der Unit wird von rechts nach links ausgewertet).

Unterstützende Klassen verwenden

Das folgende Beispiel veranschaulicht die Deklaration einer unterstützenden Klasse:

```
type
  TMyClass = class
    procedure MyProc;
    function MyFunc: Integer;
  end;

  ...

  procedure TMyClass.MyProc;
  var X: Integer;
  begin
```

```
X := MyFunc;
end;

function TMyClass.MyFunc: Integer;
begin
  ...
end;

...

type
  TMyClassHelper = class helper for TMyClass
    procedure HelloWorld;
    function MyFunc: Integer;
  end;
  ...

procedure TMyClassHelper.HelloWorld;
begin
  writeln(Self.ClassName); // Self bezieht sich auf TMyClass, nicht auf TMyClassHelper
end;

function TMyClassHelper.MyFunc: Integer;
begin
  ...
end;

...

var
  X: TMyClass;
begin
  X := TMyClass.Create;
  X.MyProc;    // Aufruf von TMyClass.MyProc
  X.HelloWorld; // Aufruf von TMyClassHelper.HelloWorld
  X.MyFunc;    // Aufruf von TMyClassHelper.MyFunc
```

Aufgerufen wird die Funktion MyFunc der unterstützenden Klasse, da diese Vorrang vor dem eigentlichen Klassentyp hat.

Siehe auch

[Klassen und Objekte](#) (siehe Seite 948)

[Felder](#) (siehe Seite 954)

[Methoden](#) (siehe Seite 956)

[Eigenschaften](#) (siehe Seite 965)

[Verschachtelte Typdeklarationen](#) (siehe Seite 982)

[Überladene Operatoren](#) (siehe Seite 984)

[Klassenreferenzen](#) (siehe Seite 974)

[Exceptions](#) (siehe Seite 977)

4.1.1.7 Standardroutinen und E/A

Dieser Abschnitt enthält eine Beschreibung der Standardroutinen in der Laufzeitbibliothek von Delphi.

4.1.1.7.1 Standardroutinen und E/A

Diese Themen behandeln die Text- und Datei-E/A und geben einen Überblick über die Standardbibliotheks Routinen. Viele der hier aufgeführten Prozeduren und Funktionen sind in den Units `System` und `SysInit` definiert, die implizit mit jeder Anwendung verwendet werden. Andere Routinen sind im Compiler integriert, werden jedoch so behandelt, als wären sie in der Unit `System` enthalten.

Einige Standardroutinen befinden sich in Units wie `SysUtils`, die in einer `uses`-Klausel aufgeführt werden müssen, wenn sie in ein Programm eingebunden werden sollen. `System` darf jedoch nicht in einer `uses`-Klausel angegeben werden. Außerdem sollten Sie die Unit `System` weder bearbeiten noch explizit neu compilieren.

Dateiein- und -ausgabe

Die folgende Tabelle enthält die Ein- und Ausgaberoutinen.

Ein- und Ausgaberoutinen

Prozedur oder Funktion	Beschreibung
Append	Öffnet eine vorhandene Textdatei zum Anhängen von Daten.
AssignFile	Weist einer Dateivariablen den Namen einer externen Datei zu.
BlockRead	Liest einen oder mehrere Blöcke aus einer untypisierten Datei.
BlockWrite	Schreibt einen oder mehrere Blöcke in eine untypisierte Datei.
ChDir	Wechselt das aktuelle Verzeichnis.
CloseFile	Schließt eine geöffnete Datei.
Eof	Gibt den EOF-Status einer Datei zurück.
Eoln	Gibt den Zeilenende-Status einer Textdatei zurück.
Erase	Löscht eine externe Datei.
FilePos	Gibt die aktuelle Position in einer typisierten oder untypisierten Datei zurück.
FileSize	Gibt die aktuelle Größe einer Datei zurück (nicht für Textdateien).
Flush	Leert den Puffer einer Ausgabe-Textdatei.
GetDir	Gibt das aktuelle Verzeichnis eines bestimmten Laufwerks zurück.
IOResult	Gibt einen Integer-Wert zurück, der den Status der zuletzt durchgeföhrten E/A-Operation angibt.
MkDir	Legt ein Unterverzeichnis an.
Read	Liest einen oder mehrere Werte aus einer Datei in eine oder mehrere Variablen.
Readln	Wie <code>Read</code> , positioniert aber anschließend den Dateizeiger auf den Anfang der nächsten Zeile der Textdatei.
Rename	Benennt eine externe Datei um.
Reset	Öffnet eine vorhandene Datei.
Rewrite	Erzeugt und öffnet eine neue Datei.
RmDir	Entfernt ein leeres Unterverzeichnis.
Seek	Setzt den Dateizeiger in einer typisierten oder untypisierten Datei auf die angegebene Position (nicht bei Textdateien).
SeekEof	Gibt den EOF-Status einer Textdatei zurück.
SeekEoln	Gibt den Zeilenende-Status einer Textdatei zurück.
SetTextBuf	Weist einer Textdatei einen E/A-Puffer zu.

Truncate	Schneidet eine typisierte oder untypisierte Datei an der aktuellen Position ab.
Write	Schreibt einen oder mehrere Werte in eine Datei.
Writeln	Wie <i>Write</i> , schreibt aber anschließend ein Zeilenendezeichen in die Textdatei.

Jede Variable vom Typ *File* ist eine Dateivariable. Es gibt drei Klassen von Dateien: typisierte Dateien, untypisierte Dateien und Textdateien. Die Syntax für die Deklaration von Dateitypen finden Sie unter "Dateitypen". Beachten Sie, dass Dateitypen nur auf der Win32-Plattform zur Verfügung stehen.

Bevor eine Dateivariable verwendet werden kann, muss sie durch einen Aufruf der Prozedur *AssignFile* einer externen Datei zugeordnet werden. Eine externe Datei ist entweder eine Datei auf einem Laufwerk oder ein Gerät (beispielsweise die Tastatur oder der Bildschirm). Die externe Datei speichert Informationen oder stellt sie zur Verfügung.

Nachdem die Zuordnung zu einer externen Datei hergestellt wurde, muss die Dateivariable *geöffnet* werden, um Ein- oder Ausgaben zu ermöglichen. Eine vorhandene Datei kann mit der Prozedur *Reset* geöffnet werden. Mit der Prozedur *Rewrite* wird eine neue Datei erzeugt und geöffnet. Textdateien, die mit der Prozedur *Reset* geöffnet wurden, erlauben nur Lesezugriffe. Textdateien, die mit *Rewrite* oder *Append* geöffnet wurden, erlauben nur Schreibzugriffe. Typisierte und untypisierte Dateien erlauben Lese- und Schreibzugriffe, unabhängig davon, ob sie mit *Reset* oder *Rewrite* geöffnet wurden.

Jede Datei enthält eine lineare Folge von Komponenten, die alle dem Komponententyp (oder dem Record-Typ) der Datei entsprechen. Jeder Komponente ist eine Nummer zugeordnet.

Die erste Komponente einer Datei hat die Nummer 0. Normalerweise erfolgt der Dateizugriff sequenziell, das heißt, beim Lesen einer Komponente mit Hilfe der Standardprozedur *Read* oder beim Schreiben mit Hilfe der Standardprozedur *Write* wird der Dateizeiger auf die nächste Komponente positioniert. Auf typisierte und untypisierte Dateien können Sie jedoch auch wahlfrei zugreifen, indem Sie die Standardprozedur *Seek* verwenden, die den Dateizeiger von der aktuellen zur angegebenen Position verschiebt. Mit den Standardfunktionen *FilePos* und *FileSize* können Sie die aktuelle Position des Dateizeigers und die aktuelle Größe der Datei ermitteln.

Nach der Bearbeitung durch das Programm muss die Datei mit der Standardprozedur *CloseFile* geschlossen werden. Nach dem Schließen wird die entsprechende externe Datei aktualisiert. Die Dateivariable kann dann einer anderen externen Datei zugeordnet werden.

In der Voreinstellung werden sämtliche Aufrufe von Standard-E/A-Routinen auf Fehler überprüft. Beim Auftreten eines Fehlers wird eine Exception ausgelöst (bzw. das Programm abgebrochen, falls keine Exception-Behandlung aktiviert ist). Die automatische Überprüfung kann mit Hilfe der Compiler-Direktiven `{$I+}` und `{$I-}` ein- und ausgeschaltet werden. Wenn die E/A-Prüfung ausgeschaltet ist (wenn also eine Routine im Status `{$I-}` compiliert wird), löst ein E/A-Fehler keine Exception aus. In diesem Fall muss zur Überprüfung des Ergebnisses einer E/A-Operation die Standardfunktion *IOResult* aufgerufen werden.

IOResult sollte auch dann aufgerufen werden, wenn der aufgetretene Fehler für das Programm nicht weiter von Bedeutung ist. Wenn Sie den Aufruf unterlassen, schlägt der nächste Aufruf einer E/A-Funktion im Status `{$I-}` fehl.

Textdateien

Dieser Abschnitt behandelt E/A-Operationen mit Dateivariablen vom Standardtyp *Text*.

Beim Öffnen einer Datei vom Typ *Text* wird die externe Datei als Folge von Zeichen interpretiert, die in Form von Zeilen vorliegen. Am Ende jeder Zeile steht ein Zeilenendezeichen (das Zeichen für Wagenrücklauf, eventuell gefolgt vom Zeichen für einen Zeilenvorschub). Der Typ *Text* unterscheidet sich vom Typ *Char*.

Für Textdateien gibt es besondere Formen der Prozeduren *Read* und *Write*, mit denen Sie Werte lesen und schreiben können, die nicht vom Typ *Char* sind. Die Werte werden automatisch in ihre Zeichendarstellung übersetzt. Die Prozedur *Read(F, I)* liest beispielsweise eine Folge von Ziffern, interpretiert sie als dezimale Integer-Werte und speichert sie in *I* (wobei *I* vom Typ *Integer* ist).

Als Standard-Textdateivariablen sind *Input* und *Output* definiert. *Input* ermöglicht nur Lesezugriffe und ist der

Standard-Eingabedatei des Betriebssystems (normalerweise die Tastatur) zugeordnet. Output ermöglicht nur Schreibzugriffe und ist der Standard-Ausgabedatei des Betriebssystems (normalerweise der Bildschirm) zugeordnet. Input und Output werden vor der Ausführung eines Programms automatisch geöffnet. Dies entspricht der Ausführung folgender Anweisungen:

```
AssignFile(Input, '');
Reset(Input);
AssignFile(Output, '');
Rewrite(Output);
```

Anmerkung: Für Win32-Anwendungen sind textorientierte E/A-Operationen nur in Konsolenanwendungen verfügbar, also in Anwendungen, die entweder mit der Option **Konsolenanwendung** (Registerkarte *Linker* im Dialogfeld *Projektoptionen*) oder mit der Befehlszeilschalter **-cc** kompiliert wurden. In einer GUI-Anwendung verursacht jede Lese- oder Schreiboperation mit Input oder Output einen E/A-Fehler.

Nicht allen Standard-E/A-Routinen, die mit Textdateien arbeiten, muss explizit eine Dateivariable als Parameter übergeben werden. Wenn der Parameter fehlt, wird standardmäßig Input oder Output verwendet, je nachdem, ob die Prozedur oder Funktion eingabe- oder ausgabeorientiert ist. `Read(X)` entspricht beispielsweise `Read(Input, X)`, und `Write(X)` entspricht `Write(Output, X)`.

Eine Datei, die einer Ein- oder Ausgabерoutine für Textdateien übergeben wird, muss zuvor mit `AssignFile` einer externen Datei zugeordnet und mit `Reset`, `Rewrite` oder `Append` geöffnet werden. Wenn Sie eine mit `Reset` geöffnete Datei an eine ausgabeorientierte Prozedur oder Funktion übergeben, tritt ein Fehler auf. Gleiches gilt, wenn Sie eine mit `Rewrite` oder `Append` geöffnete Datei an eine eingabeorientierte Prozedur übergeben.

Untypisierte Dateien

Untypisierte Dateien kann man sich als Low-Level-E/A-Kanäle vorstellen, die vorrangig für den direkten Zugriff auf Dateien verwendet werden, und zwar unabhängig von deren Typ und Struktur. Eine untypisierte Datei wird nur mit dem Wort **file** deklariert. Beispiel:

```
var DataFile: file;
```

Bei untypisierten Dateien kann den Prozeduren `Reset` und `Rewrite` ein Parameter übergeben werden, der die Größe der Blöcke bei Lese- und Schreiboperationen festlegt. Die Standardgröße beträgt aus historischen Gründen 128 Byte. Nur der Wert 1 führt zuverlässig bei jeder beliebigen Datei zur richtigen Größe (ein Block der Größe 1 kann nicht weiter unterteilt werden).

Mit Ausnahme von `Read` und `Write` können alle Standardroutinen für typisierte Dateien auch für untypisierte Dateien verwendet werden. Anstelle von `Read` und `Write` stehen mit `BlockRead` und `BlockWrite` zwei Prozeduren für besonders schnelle Lese- und Schreiboperationen zur Verfügung.

Gerätetreiber für Textdateien

Sie können für Programme Ihre eigenen Gerätetreiber für Textdateien definieren. Ein Gerätetreiber für Textdateien besteht aus vier Funktionen, die ein Interface zwischen einem Gerät und dem Dateisystem von Delphi implementieren.

Jeder Gerätetreiber wird durch die Funktionen `Open`, `InOut`, `Flush` und `Close` definiert. Der Kopf einer jeden Funktion wird folgendermaßen deklariert:

```
function DeviceFunc(var F: TTextRec): Integer;
```

Dabei ist `DeviceFunc` der Name der entsprechenden Funktion (d. h. `Open`, `InOut`, `Flush` oder `Close`). Der Rückgabewert einer Gerätetreiberfunktion ist immer der von `IOResult` zurückgegebene Wert. Wenn 0 zurückgegeben wird, war die Operation erfolgreich.

Um die Gerätetreiberfunktionen einer bestimmten Datei zuzuordnen, müssen Sie selbst eine eigene `Assign`-Prozedur schreiben. Diese Prozedur muss den vier Funktionszeigern der Textdateivariablen die Adressen der vier Gerätetreiberfunktionen zuweisen. Zusätzlich sollte sie dem Feld `Mode` die Konstante `fmClosed`, dem Feld `BufSize` die Größe des Textdateipuffers, dem Feld `BufPtr` einen Zeiger auf den Textdateipuffer und dem Feld `Name` einen Leer-String zuweisen.

Mit den vier Gerätetreiberfunktionen `DevOpen`, `DevInOut`, `DevFlush` und `DevClose` könnte die Prozedur `Assign`

beispielsweise wie folgt aussehen:

```
procedure AssignDev(var F: Text);
begin
  with TTextRec(F) do
  begin
    Mode := fmClosed;
    BufSize := SizeOf(Buffer);
    BufPtr := @Buffer;
    OpenFunc := @DevOpen;
    InOutFunc := @DevInOut;
    FlushFunc := @DevFlush;
    CloseFunc := @DevClose;
    Name[0] := #0;
  end;
end;
```

Die Gerätetreiberfunktionen können das Feld *UserData* im Datei-Record zum Speichern interner Informationen verwenden, da dieses Feld vom produktsspezifischen Dateisystem nie geändert wird.

Die Funktion Open

Die Funktion *Open* wird von den Standardprozeduren *Reset*, *Rewrite* und *Append* zum Öffnen von Textdateien verwendet, die einem Gerät zugeordnet sind. Das Feld *Mode* enthält beim Aufruf einen der Werte *fmlInput*, *fmOutput* oder *fmlInOut*, der angibt, ob *Open* von *Reset*, *Rewrite* oder *Append* aufgerufen wurde.

Open bereitet die Datei entsprechend dem Wert von *Mode* für die Ein- oder Ausgabe vor. Wenn *Mode* den Wert *fmlInOut* hat (was bedeutet, dass *Open* von *Append* aufgerufen wurde), muss dieser in *fmOutput* geändert werden, bevor *Open* beendet wird.

Die Funktion *Open* wird immer vor allen anderen Gerätetreiberfunktionen aufgerufen. Aus diesem Grund initialisiert *AssignDev* nur das Feld *OpenFunc* und überlässt die Initialisierung der restlichen Felder der Funktion *Open*. Je nach dem Wert von *Mode* kann *Open* dann die Zeiger für die ein- oder ausgabeorientierten Funktionen installieren. Die Funktionen *InOut* und *Flush* und die Prozedur *CloseFile* brauchen also den aktuellen Modus nicht zu ermitteln.

Die Funktion InOut

Die Funktion *InOut* wird von den Standardroutinen *Read*, *ReadLn*, *Write*, *Writeln*, *Eof*, *Eoln*, *SeekEof*, *SeekEoln* und *CloseFile* immer dann aufgerufen, wenn eine Ein- oder Ausgabeoperation ansteht.

Wenn *Mode* den Wert *fmlInput* hat, liest die Funktion *InOut* maximal *BufSize* Zeichen in *BufPtr* und gibt in *BufEnd* die Anzahl der gelesenen Zeichen zurück. Außerdem wird *BufPos* der Wert 0 zugewiesen. Wenn die Funktion *InOut* als Ergebnis einer Eingabeaufforderung in *BufEnd* den Wert 0 zurückgibt, ist *Eof* für diese Datei **True**.

Wenn *Mode* auf *fmOutput* gesetzt ist, schreibt die Funktion *InOut* *BufPos* Zeichen aus *BufPtr* und gibt in *BufPos* den Wert 0 zurück.

Die Funktion Flush

Die Funktion *Flush* wird nach jedem *Read*, *ReadLn*, *Write* und *Writeln* aufgerufen. Sie leert optional den Textdateipuffer.

Wenn *Mode* auf *fmlInput* gesetzt ist, kann *Flush* in *BufPos* und *BufEnd* den Wert 0 speichern, um die verbleibenden (nicht gelesenen) Zeichen im Puffer zu löschen. Diese Möglichkeit wird aber nur selten verwendet.

Wenn *Mode* auf *fmOutput* gesetzt ist, kann *Flush* wie die Funktion *InOut* den Inhalt des Puffers schreiben. Dadurch wird sichergestellt, dass der geschriebene Text sofort angezeigt wird. Wenn *Flush* keine Aktion durchführt, wird der Text erst dann auf dem Gerät angezeigt, wenn der Puffer voll ist oder die Datei geschlossen wird.

Die Funktion Close

Die Funktion *Close* wird von der Standardprozedur *CloseFile* aufgerufen. Sie schließt eine Textdatei, die einem Gerät zugeordnet ist. Die Prozeduren *Reset*, *Rewrite* und *Append* rufen *Close* ebenfalls auf, wenn die zu öffnende Datei bereits

geöffnet ist. Wenn Mode auf fmOutput gesetzt ist, ruft das Dateisystem vor dem Aufruf von Close die Funktion InOut auf, um sicherzustellen, dass alle Zeichen auf das Gerät geschrieben wurden.

Nullterminierte Strings

Aufgrund der erweiterten Syntax von Delphi können die Standardprozeduren Read, Readln, Str und Val mit nullbasierten Zeichen-Arrays und die Standardprozeduren Write, Writeln, Val, AssignFile und Rename sowohl mit nullbasierten Zeichen-Arrays als auch mit Zeichenzeigern umgehen.

Funktionen für nulterminierte Strings

Die folgenden Funktionen zur Bearbeitung nulterminierter Strings stehen zur Verfügung.

Funktionen für nulterminierte Strings

Funktion	Beschreibung
StrAlloc	Reserviert auf dem Heap einen Zeichenpuffer der angegebenen Größe.
StrBufSize	Gibt die Größe eines Zeichenpuffers zurück, der mit StrAlloc oder StrNew angelegt wurde.
StrCat	Verkettet zwei Strings.
StrComp	Vergleicht zwei Strings.
StrCopy	Kopiert einen String.
StrDispose	Gibt einen Zeichenpuffer frei, der mit StrAlloc oder StrNew angelegt wurde.
StrECopy	Kopiert einen String und gibt einen Zeiger auf das Ende des Strings zurück.
StrEnd	Gibt einen Zeiger auf das Ende eines Strings zurück.
StrFmt	Formatiert einen oder mehrere Werte in einem String.
StrICmp	Vergleicht zwei Strings ohne Berücksichtigung der Groß-/Kleinschreibung.
StrLCat	Verkettet zwei Strings bei vorgegebener Maximallänge des resultierenden Strings.
StrLComp	Vergleicht die vorgegebene Maximallänge zweier Strings.
StrLCopy	Kopiert einen String bis zu einer vorgegebenen Maximallänge.
StrLen	Gibt die Länge eines Strings zurück.
StrLFmt	Formatiert einen oder mehrere Werte zur Positionierung in einem String mit vorgegebener Maximallänge.
StrLICmp	Vergleicht die Länge von zwei Strings ohne Beachtung der Groß-/Kleinschreibung.
StrLower	Konvertiert einen String in Kleinbuchstaben.
StrMove	Verschiebt eine Gruppe von Zeichen aus einem String in einen anderen.
StrNew	Weist Speicherplatz für einen String auf dem Heap zu.
StrPCopy	Kopiert einen Pascal-String in einen nulterminierten String.
StrPLCopy	Kopiert einen Pascal-String in einen nulterminierten String mit vorgegebener Länge.
StrPos	Gibt einen Zeiger auf das erste Vorkommen eines bestimmten Teilstrings innerhalb eines Strings zurück.
StrRScan	Gibt einen Zeiger auf das letzte Vorkommen eines bestimmten Zeichens innerhalb eines Strings zurück.
StrScan	Gibt einen Zeiger auf das erste Vorkommen eines bestimmten Zeichens innerhalb eines Strings zurück.
StrUpper	Konvertiert einen String in Großbuchstaben.

Für die Standardfunktionen zur String-Bearbeitung gibt es jeweils Gegenstücke, die Multibyte-Zeichensätze unterstützen und die sprachspezifische Sortierfolge für Zeichen implementieren. Die Namen der Multibyte-Funktionen beginnen mit *Ansi*. So ist *AnsiStrPos* beispielsweise die Multibyte-Version von *StrPos*. Die Unterstützung von Multibyte-Zeichen ist betriebssystemabhängig und basiert auf dem verwendeten Gebietsschema.

WideStrings

Die Unit System stellt drei Funktionen zur Verfügung, die zur Umwandlung von nullterminierten WideStringen in lange Einzel- oder Doppelbyte-Strings verwendet werden können: WideCharToString, WideCharLenToString und StringToWideChar.

Durch Zuweisung kann ebenfalls zwischen Strings konvertiert werden. So sind beispielsweise die folgenden beiden Aussagen gültig:

```
MyAnsiString := MyWideString;
MyWideString := MyAnsiString;
```

Weitere Standardroutinen

In der folgenden Tabelle finden Sie einige (aber nicht alle) der gebräuchlichsten Prozeduren und Funktionen aus den CodeGear-Produktbibliotheken.

Weitere Standardroutinen

Prozedur oder Funktion	Beschreibung
Addr	Gibt einen Zeiger auf ein bestimmtes Objekt zurück.
AllocMem	Weist einen Speicherblock zu und initialisiert jedes Byte mit Null.
ArcTan	Berechnet den Arcustangens der angegebenen Zahl.
Assert	Löst eine Exception aus, wenn der übergebene Ausdruck nicht zu True ausgewertet werden kann.
Assigned	Überprüft einen Zeiger oder eine prozedurale Variable auf nil .
Beep	Generiert einen Standard-Signalton.
Break	Beendet eine for -, while - oder repeat -Anweisung vorzeitig.
ByteToCharIndex	Gibt die Position eines Zeichens in einem String zurück, das ein bestimmtes Byte enthält.
Chr	Gibt das Zeichen mit dem angegebenen Integer-Wert zurück.
Close	Schließt eine Datei.
CompareMem	Führt einen binären Vergleich zweier Speicherabbilder durch.
CompareStr	Vergleicht Strings unter Berücksichtigung der Groß-/Kleinschreibung.
CompareText	Vergleicht Strings auf der Grundlage ihrer Ordinalwerte ohne Berücksichtigung der Groß-/Kleinschreibung.
Continue	Führt die nächste Iteration einer for -, while - oder repeat -Anweisung aus.
Copy	Gibt einen Teilstring eines Strings oder ein Segment eines dynamischen Arrays zurück.
Cos	Berechnet den Cosinus eines Winkels.
CurrToStr	Konvertiert einen Währungswert in einen String.
Date	Gibt das aktuelle Datum zurück.
DateTimeToStr	Konvertiert eine Variable des Typs <i>TDateTime</i> in einen String.
DateToStr	Konvertiert eine Variable des Typs <i>TDateTime</i> in einen String.
Dec	Erniedrigt eine ordinale Variable oder eine typisierte Zeigervariable.
Dispose	Gibt dynamisch zugewiesenen Variablenspeicher frei.
ExceptAddr	Gibt die Adresse zurück, unter der die aktuelle Exception ausgelöst wurde.
Exit	Beendet die aktuelle Prozedur.
Exp	Berechnet den Exponenten von X.

FillChar	Füllt einen Block aufeinander folgender Bytes mit einem bestimmten Wert.
Finalize	Deinitialisiert eine dynamisch zugewiesene Variable.
FloatToStr	Konvertiert eine Gleitkommazahl in den entsprechenden String-Wert.
FloatToStrF	Konvertiert einen Gleitkommawert in einen String mit einem bestimmten Format.
FmtLoadStr	Gibt die Ausgabe von Format für einen bestimmten String zurück.
FmtStr	Gibt einen formatierten String zurück, der aus einer Reihe von Array-Argumenten gebildet wird.
Format	Gibt einen formatierten String zurück, der aus einem Format-String und einer Reihe von Array-Argumenten gebildet wird.
FormatDateTime	Formatiert einen Datums-/Zeitwert.
FormatFloat	Formatiert einen Gleitkommawert.
FreeMem	Gibt zugewiesenen Speicher frei.
GetMem	Weist dynamischen Speicher und einen Zeiger auf die Adresse des Blocks zu.
Halt	Breicht ein Programm ab.
Hi	Gibt das höherwertige Byte eines Ausdrucks als vorzeichenlosen Wert zurück.
High	Gibt den höchsten Wert im Bereich eines Typs, Arrays oder Strings zurück.
Inc	Erhöht eine ordinale Variable oder eine typisierte Zeigervariable.
Initialize	Initialisiert eine dynamisch zugewiesene Variable.
Insert	Fügt einen Teilstring an der angegebenen Position in einen String ein.
Int	Gibt den ganzzahligen Anteil einer reellen Zahl zurück.
IntToStr	Konvertiert einen Integer-Wert in einen String.
Length	Gibt die Länge eines Strings oder die Größe eines Arrays zurück.
Lo	Gibt das niederwertige Byte eines Ausdrucks als vorzeichenlosen Wert zurück.
Low	Gibt den niedrigsten Wert im Bereich eines Typs, Arrays oder Strings zurück.
LowerCase	Wandelt einen ASCII-String in Kleinbuchstaben um.
MaxIntValue	Gibt den größten vorzeichenbehafteten Wert in einem Integer-Array zurück.
MaxValue	Gibt den größten vorzeichenbehafteten Wert in einem Array zurück.
MinIntValue	Liefert den kleinsten vorzeichenbehafteten Wert eines Integer-Arrays zurück.
MinValue	Gibt den kleinsten vorzeichenbehafteten Wert eines Arrays zurück.
Neu	Erzeugt einen dynamisch zuweisbaren Variablen Speicher und referenziert diesen über den angegebenen Zeiger.
Now	Gibt das aktuelle Datum und die aktuelle Uhrzeit zurück.
Ord	Gibt den ordinalen Integerwert eines Ausdrucks mit ordinalem Typ zurück.
Pos	Gibt den Index des ersten Einzelbyte-Zeichens eines angegebenen Teilstrings innerhalb eines Strings zurück.
Pred	Gibt den Vorgänger eines ordinalen Wertes zurück.
Ptr	Konvertiert einen Wert in einen Zeiger.
Random	Erzeugt eine Zufallszahl innerhalb eines bestimmten Bereichs.
ReallocMem	Weist einen dynamisch zuweisbaren Speicher neu zu.
Round	Rundet eine reelle Zahl auf die nächste ganze Zahl.
SetLength	Legt die dynamische Länge einer String-Variablen oder eines Arrays fest.

SetString	Setzt Inhalt und Länge eines Strings.
ShowException	Zeigt eine Exception-Meldung und ihre physikalische Adresse an.
ShowMessage	Zeigt ein Meldungsfenster mit einem unformatierten String und der Schaltfläche OK an.
ShowMessageFmt	Zeigt ein Meldungsfenster mit einem formatierten String und der Schaltfläche OK an.
Sin	Gibt den Sinus eines Winkels zurück.
SizeOf	Gibt die Anzahl der von einer Variablen oder einem Typ belegten Bytes zurück.
Sqr	Gibt das Quadrat eines Wertes zurück.
Sqrt	Gibt die Quadratwurzel einer Zahl zurück.
Str	Konvertiert einen Integer- oder Real-Wert in einen String.
StrToCurr	Konvertiert einen String in einen Währungswert.
StrToDate	Konvertiert einen String in einen Datums Wert (<i>TDateTime</i> -Objekt).
StrToDateTime	Konvertiert einen String in einen <i>TDateTime</i> -Wert.
StrToFloat	Konvertiert einen String in einen Gleitkommawert.
StrToInt	Konvertiert einen String in einen Integer-Wert.
StrToTime	Konvertiert einen String in ein <i>TDateTime</i> -Objekt.
StrUpper	Gibt einen ASCII-String in Großbuchstaben zurück.
Succ	Gibt den Nachfolger einer Ordinalzahl zurück.
Sum	Berechnet die Summe aller Elemente eines Arrays.
Time	Gibt die aktuelle Uhrzeit zurück.
TimeToStr	Konvertiert eine Variable des Typs <i>TDateTime</i> in einen String.
Trunc	Konvertiert eine reelle Zahl in einen Integer-Wert.
UniqueString	Stellt sicher, dass ein String nur eine Referenz hat (der String kann kopiert werden, um eine einzelne Referenz zu erzeugen).
UpCase	Konvertiert ein Zeichen in einen Großbuchstaben.
UpperCase	Gibt einen String in Großbuchstaben zurück.
VarArrayCreate	Erstellt ein variantes Array.
VarArrayDimCount	Gibt die Anzahl der Dimensionen eines varianten Arrays zurück.
VarArrayHighBound	Gibt die Obergrenze einer Dimension in einem varianten Array zurück.
VarArrayLock	Sperrt ein variantes Array und gibt einen Zeiger auf die Daten zurück.
VarArrayLowBound	Gibt die Untergrenze einer Dimension in einem varianten Array zurück.
VarArrayOf	Erstellt und füllt ein eindimensionales, variantes Array.
VarArrayRedim	Ändert die Größe eines varianten Arrays.
VarArrayRef	Gibt eine Referenz auf das übergebene variante Array zurück.
VarArrayUnlock	Entsperrt ein variantes Array.
VarAsType	Konvertiert eine Variante in den angegebenen Typ.
VarCast	Konvertiert eine Variante in den angegebenen Datentyp und speichert das Ergebnis in einer Variablen.
VarClear	Löscht eine Variante.
VarCopy	Kopiert eine Variante.
VarToStr	Konvertiert eine Variante in einen String.

VarType	Gibt den Typencode der angegebenen Variante zurück.
---------	---

Siehe auch

Datentypen (siehe Seite 887)

VCL-Anwendungen in Delphi für .NET portieren

4.1.1.8 Bibliotheken und Packages

Dieser Abschnitt enthält Informationen zur Erstellung statischer und dynamisch ladbarer Bibliotheken in Delphi.

4.1.1.8.1 Bibliotheken und Packages

Unter einer dynamisch ladbaren Bibliothek versteht man in Win32 eine dynamische Linkbibliothek ([DLL](#)) und in .NET eine Assemblierung (ebenfalls eine [DLL](#)). Es handelt sich dabei um eine Sammlung von Routinen, die von Anwendungen und von anderen DLLs bzw. gemeinsamen Objekten aufgerufen werden können. Dynamisch ladbare Bibliotheken enthalten wie Units gemeinsam genutzten Code und Ressourcen. Sie stellen jedoch eine separat compilierte, ausführbare Datei dar, die zur Laufzeit zu den Programmen gelinkt wird, die sie verwenden.

Delphi-Programme können DLLs und Assemblierungen aufrufen, die in anderen Sprachen geschrieben wurden. Anwendungen, die in anderen Sprachen programmiert wurden, können DLLs oder Assemblierungen aufrufen, die mit Delphi erstellt wurden.

Dynamisch ladbare Bibliotheken aufrufen

Direkt aufgerufene Betriebssystemroutinen werden erst zur Laufzeit zur Anwendung gelinkt. Das bedeutet, dass die Bibliothek zur Compilierzeit nicht benötigt wird. Es bedeutet aber auch, dass der Compiler nicht überprüfen kann, ob eine Routine korrekt importiert wird.

Bevor Sie Routinen aufrufen können, die in einer DLL oder Assemblierung definiert sind, müssen Sie diese Routinen importieren. Dies kann auf zwei Arten erfolgen: durch Deklarieren einer Prozedur oder Funktion als **external** oder durch direkte Betriebssystemaufrufe. Bei beiden Methoden werden die Routinen erst zur Laufzeit zur Anwendung gelinkt.

Variablen aus DLLs oder Assemblierungen können in Delphi nicht importiert werden.

Statisches Laden

Die einfachste Art, eine Prozedur oder Funktion zu importieren, besteht darin, sie mit der Direktive **external** zu deklarieren. Zum Beispiel:

```
procedure DoSomething; external 'MYLIB.DLL';
```

Durch diese Deklaration wird beim Programmstart die Datei [MYLIB.DLL](#) geladen. Während der Programmausführung bezieht sich der Bezeichner `DoSomething` immer auf denselben Eintrittspunkt in derselben gemeinsamen Bibliothek.

Sie können die Deklaration einer importierten Routine direkt in das Programm oder die Unit einfügen, in dem bzw. der sie aufgerufen wird. Um Ihre Programme leichter pflegen zu können, sollten Sie aber alle **external**-Deklarationen in einer separaten "Import-Unit" zusammenfassen. Diese Unit enthält dann auch die Konstanten und Typen, die für das Interface zur Bibliothek erforderlich sind. Andere Module, die auf die Import-Unit zugreifen, können alle darin deklarierten Routinen aufrufen.

Dynamisches Laden

Sie können auf die Routinen einer Bibliothek über direkte Aufrufe von Betriebssystemfunktionen zugreifen (z. B. `LoadLibrary`, `FreeLibrary` und `GetProcAddress`). Diese Funktionen sind in der Datei [Windows.pas](#) deklariert.

Zum Beispiel:

```
uses Windows, ...;
```

```

4

type
  TTimeRec = record
    Second: Integer;
    Minute: Integer;
    Hour: Integer;
  end;

TGetTime = procedure(var Time: TTimeRec);
THandle = Integer;

var
  Time: TTimeRec;
  Handle: THandle;
  GetTime: TGetTime;
  .

.

begin
  Handle := LoadLibrary('libraryname');
  if Handle <> 0 then
  begin
    @GetTime := GetProcAddress(Handle, 'GetTime');
    if @GetTime <> nil then
    begin
      GetTime(Time);
      with Time do
        WriteLn('Es ist jetzt ', Hour, ':', Minute, ':', Second);
    end;
    FreeLibrary(Handle);
  end;
end;

```

Wenn Sie Routinen auf diese Weise importieren, wird die Bibliothek erst bei der Ausführung des Quelltextes geladen, der den Aufruf von `LoadLibrary` enthält. Später wird die Bibliothek mit einem Aufruf von `FreeLibrary` wieder aus dem Speicher entfernt. Auf diese Weise wird Speicherplatz gespart und das Programm auch dann ausgeführt, wenn einige der aufgerufenen Bibliotheken nicht zur Verfügung stehen.

Siehe auch

[Dynamisch ladbare Bibliotheken schreiben](#) (siehe Seite 998)

[Packages](#) (siehe Seite 1002)

4.1.1.8.2 Dynamisch ladbare Bibliotheken schreiben

In den folgenden Abschnitten werden die Elemente beschrieben, die bei der Erstellung einer dynamisch ladbaren Bibliothek zum Einsatz kommen:

- Die `exports`-Klausel
- Code für die Initialisierung der Bibliothek
- Globale Variablen
- Bibliotheken und Systemvariablen

Die `exports`-Klausel

Die Grundstruktur einer dynamisch ladbaren Bibliothek ist identisch mit der eines Programms. Der einzige Unterschied besteht darin, dass sie nicht mit `program`, sondern mit dem reservierten Wort `library` beginnt.

Andere Bibliotheken und Programme können nur auf die Routinen zugreifen, die eine Bibliothek explizit exportiert. Das folgende Beispiel zeigt eine Bibliothek mit den exportierten Funktionen `Min` und `Max`:

```

library MinMax;
  function Min(X, Y: Integer): Integer; stdcall;
begin
  if X < Y then Min := X else Min := Y;
end;
function Max(X, Y: Integer): Integer; stdcall;
begin
  if X > Y then Max := X else Max := Y;
end;
exports
  Min,
  Max;
begin
end.

```

Sie können eine Bibliothek auch Anwendungen zur Verfügung stellen, die in anderen Sprachen geschrieben wurden. Für diesen Zweck ist es am sichersten, wenn Sie in den Deklarationen exportierter Funktionen die Direktive **stdcall** angeben. Die standardmäßig verwendete Delphi-Aufrufkonvention **register** wird nicht von allen anderen Sprachen unterstützt.

Bibliotheken können aus mehreren Units bestehen. In diesem Fall enthält die Quelltextdatei der Bibliothek nur eine **uses**-Klausel, eine **exports**-Klausel und den Initialisierungscode. Zum Beispiel:

```

library Editors;
uses EdInit, EdInOut, EdFormat, EdPrint;
exports
  InitEditors,
  DoneEditors name Done,
  InsertText name Insert,
  DeleteSelection name Delete,
  FormatSelection,
  PrintSelection name Print,
  .
  .
  .
  SetErrorHandler;
begin
  InitLibrary;
end.

```

Die **exports**-Klauseln können in den **interface**- oder **implementation**-Abschnitt einer Unit eingefügt werden. Alle Bibliotheken mit einer solchen Unit in ihrer **uses**-Klausel exportieren automatisch die Routinen, die in der **exports**-Klausel der Unit aufgelistet sind, ohne dass eine eigene **exports**-Klausel benötigt wird.

Die Direktive **local**, die Routinen als nicht verfügbar für den Export markiert, ist plattformspezifisch und hat keinen Einfluss auf die Windows-Programmierung.

function Contraband(I: Integer): Integer; local;

**** Eine Routine wird exportiert, wenn sie in einer **exports**-Klausel wie der folgenden angegeben wird:

exports Eintritt1, ..., Eintrittn;

Eintritt steht für den Namen einer Prozedur, Funktion oder Variable, die vor der **exports**-Klausel deklariert werden muss. Darauf folgt eine Parameterliste (nur beim Exportieren einer überladenen Routine) und ein optionaler **name**-Bezeichner. Der Name der Prozedur oder Funktion kann mit dem Namen einer Unit qualifiziert werden.

(Eintrittspunkte können außerdem die Direktive **resident** enthalten, die der Abwärtskompatibilität dient und vom Compiler ignoriert wird.)

Der Bezeichner **index** (nur Win32) besteht aus der Direktive **index** und einer numerischen Konstante von 1 bis 2.147.483.647 (je niedriger die Konstante, desto effizienter das Programm). Ist kein **index**-Bezeichner angegeben, wird der Routine in der Exporttabelle automatisch eine Nummer zugewiesen.

Anmerkung: Der Bezeichner **index** dient lediglich der Abwärtskompatibilität und sollte nach Möglichkeit nicht verwendet werden, da er in anderen Entwicklungstools zu Problemen führen kann.

Ein **name**-Bezeichner besteht aus der Direktive **name** und einer nachfolgenden String-Konstante. Verfügt ein Eintrittspunkt über **keinen****name**-Bezeichner, wird die Routine unter ihrem ursprünglich deklarierten Namen (in derselben Schreibweise) exportiert. Verwenden Sie die **name**-Klausel, wenn Sie eine Routine unter einem anderen Namen exportieren wollen. Zum Beispiel:

```
exports
DoSomethingABC name 'DoSomething';
```

Wenn eine überladene Funktion oder Prozedur aus einer dynamisch ladbaren Bibliothek exportiert wird, muss die **exports**-Klausel die zugehörige Parameterliste enthalten. Zum Beispiel:

```
exports
Divide(X, Y: Integer) name 'Divide_Ints',
Divide(X, Y: Real) name 'Divide_Reals';
```

In Win32-Programmen dürfen überladene Routinen keine **index**-Bezeichner enthalten.

Die **exports**-Klausel kann im Deklarationsteil des Programms oder der Bibliothek bzw. im **interface**- oder **implementation**-Abschnitt einer Unit an beliebigen Stellen und beliebig oft angegeben werden. Programme enthalten nur selten eine **exports**-Klausel.

Code für die Initialisierung der Bibliothek

Die Anweisungen im Block einer Bibliothek bilden den Initialisierungscode der Bibliothek. Diese Anweisungen werden nur einmal beim Laden der Bibliothek ausgeführt. Mit diesen Anweisungen werden beispielsweise Fensterklassen registriert und Variablen initialisiert. Außerdem kann der Initialisierungscode einer Bibliothek mithilfe der Variablen **DllProc** eine Eintrittspunkt-Prozedur installieren. Die Variable **DllProc** ist einer Exit-Prozedur ähnlich, die im Abschnitt "Exit-Prozeduren" beschrieben wird. Die Eintrittspunkt-Prozedur wird beim Laden oder Entladen der Bibliothek ausgeführt.

Der Initialisierungscode einer Bibliothek kann einen Fehler signalisieren. Zu diesem Zweck wird der Variablen **ExitCode** ein Wert zugewiesen, der ungleich Null ist. **ExitCode** ist in der Unit **System** deklariert und hat den Standardwert Null. Dieser Wert gibt an, dass die Initialisierung erfolgreich war. Wenn der Initialisierungscode der Bibliothek der Variablen **ExitCode** einen anderen Wert zuweist, wird die Bibliothek aus dem Speicher entfernt, und die aufrufende Anwendung wird über den Fehler benachrichtigt. Tritt während der Ausführung des Initialisierungscodes eine nicht verarbeitete Exception auf, wird die aufrufende Anwendung über einen Fehler beim Laden der Bibliothek benachrichtigt.

Hier ein Beispiel für eine Bibliothek mit Initialisierungscode und einer Eintrittspunkt-Prozedur:

```
library Test;
var
  SaveDllProc: Pointer;
  procedure LibExit(Reason: Integer);
begin
  if Reason = DLL_PROCESS_DETACH then
    begin
      . . . // Exit-Code für Bibliothek
    end;
  SaveDllProc(Reason); // Speichern der Eintrittspunkt-Prozedur
end;
begin
  . . . // Code für die Initialisierung der Bibliothek
  SaveDllProc := DllProc; // Speichern der Kette von Exit-Prozeduren
  DllProc := @LibExit; // Installieren der LibExit-Exit-Prozedur
end.
```

DllProc wird immer dann aufgerufen, wenn die Bibliothek das erste Mal in den Speicher geladen wird, wenn ein Thread

gestartet oder angehalten wird oder wenn die Bibliothek aus dem Speicher entfernt wird. Die Initialisierungsteile aller von einer Bibliothek verwendeten Units werden vor dem Initialisierungscode der Bibliothek, die **finalization**-Abschnitte nach der Eintrittspunkt-Prozedur der Bibliothek ausgeführt.

Globale Variablen in einer Bibliothek

In einer gemeinsamen Bibliothek deklarierte globale Variablen können von Delphi-Anwendungen nicht importiert werden.

Eine Bibliothek kann von mehreren Anwendungen gleichzeitig verwendet werden. Jede Anwendung verfügt aber in ihrem Verarbeitungsbereich über eine Kopie der Bibliothek mit einem eigenen Satz globaler Variablen. Damit mehrere Bibliotheken (oder mehrere Instanzen einer Bibliothek) den Speicher gemeinsam nutzen können, müssen Speicherzuordnungstabellen verwendet werden. Weitere Informationen zu diesem Thema finden Sie in der Systemdokumentation.

Bibliotheken und Systemvariablen

Einige der in der Unit `System` deklarierten Variablen sind für Programmierer von Bibliotheken von besonderem Interesse. Mit `IsLibrary` können Sie feststellen, ob der Code in einer Anwendung oder in einer Bibliothek ausgeführt wird. `IsLibrary` ist in einer Anwendung immer **False**, in einer Bibliothek dagegen immer **True**. Während der Lebensdauer einer Bibliothek enthält die Variable `HInstance` das Instanzen-Handle der Bibliothek. Die Variable `CmdLine` ist in einer Bibliothek immer **nil**.

Die Variable `DllProc` ermöglicht einer Bibliothek die Überwachung der Betriebssystemaufrufe an ihrem Eintrittspunkt. Diese Möglichkeit wird normalerweise nur von Bibliotheken verwendet, die Multithreading unterstützen. `DllProc` kann in Windows- und Linux-Programmen verwendet werden, erfüllt dabei aber unterschiedliche Funktionen. In Win32 wird `DLLProc` in Multithread-Anwendungen eingesetzt. Es empfiehlt sich, alle Abläufe bei der Programmbeendigung nicht über Exit-Prozeduren, sondern über **finalization**-Abschnitte zu steuern.

Für die Überwachung von Betriebssystemaufrufen erstellen Sie eine Callback-Prozedur mit einem Integer-Parameter. Beispiel:

```
procedure DLLHandler(Reason: Integer);
```

Außerdem müssen Sie der Variable `DLLProc` die Adresse der Prozedur zuweisen. Beim Aufruf der Prozedur wird ihr einer der folgenden Werte übergeben:

<code>DLL_PROCESS_DETACH</code>	Gibt an, dass die Bibliothek als Ergebnis einer Beendigungsprozedur oder eines Aufrufs von <code>FreeLibrary</code> vom Adressraum des aufrufenden Prozesses getrennt wird.
<code>DLL_PROCESS_ATTACH</code>	Gibt an, dass die Bibliothek eine Verbindung mit dem Adressraum des aufrufenden Prozesses herstellt. Dies ergibt sich aus einem Aufruf von <code>LoadLibrary</code> .
<code>DLL_THREAD_ATTACH</code>	Gibt an, dass der aktuelle Prozess einen neuen Thread erstellt.
<code>DLL_THREAD_DETACH</code>	Gibt an, dass ein Thread ohne Probleme beendet wurde.

Sie können die Aktionen im Rumpf der Prozedur davon abhängig machen, welcher Parameter an die Prozedur übergeben wird.

Exceptions und Laufzeitfehler in Bibliotheken

Wenn in einer dynamisch ladbaren Bibliothek eine Exception erzeugt, aber nicht behandelt wird, wird sie nach außen an den Aufrufer weitergegeben. Wenn die aufrufende Anwendung oder Bibliothek in Delphi geschrieben wurde, kann die Exception in einer normalen `try...except`-Anweisung behandelt werden.

In Win32 kann die Exception wie eine Betriebssystem-Exception mit dem Code `$0EEDFADE` behandelt werden, wenn die aufrufende Anwendung oder Bibliothek in einer anderen Programmiersprache entwickelt wurde. Der erste Eintrag im Array `ExceptionInformation` des Records mit der Betriebssystem-Exception enthält die Exception-Adresse, der zweite Eintrag eine Referenz auf das Exception-Objekt von Delphi.

Normalerweise sollten Exceptions innerhalb der Bibliothek behandelt werden. Delphi-Exceptions entsprechen dem Exception-Modell des Betriebssystems (und dem Exception-Modell von .NET).

Wenn in einer Bibliothek die Unit `SysUtils` nicht verwendet wird, ist die Unterstützung von Exceptions deaktiviert. Tritt in diesem Fall in der Bibliothek ein Laufzeitfehler auf, wird die aufrufende Anwendung beendet. Da die Bibliothek nicht feststellen

kann, ob sie von einem Delphi-Programm aufgerufen wurde, kann sie auch nicht die Exit-Prozeduren der Anwendung aufrufen. Die Anwendung wird einfach beendet und aus dem Speicher entfernt.

Der Shared-Memory-Manager (nur Win32)

Wenn eine DLL in Win32 Routinen exportiert, die lange Strings oder dynamische Arrays als Parameter oder als Funktionsergebnis übergeben (entweder direkt oder in Records bzw. Objekten), müssen die DLL und ihre Client-Anwendungen (oder DLLs) die Unit **ShareMem** verwenden. Dasselbe gilt, wenn eine Anwendung oder DLL mit `New` oder `GetMem` Speicherplatz reserviert, der in einem anderen Modul durch einen Aufruf von **Dispose** oder **FreeMem** wieder freigegeben wird. **ShareMem** sollte in der **uses**-Klausel der Programme oder Bibliotheken, von denen sie eingebunden wird, immer an erster Stelle stehen.

ShareMem ist die Interface-Unit für den Speichermanager [BORLANDMM.DLL](#), der es Modulen ermöglicht, dynamisch zugewiesenen Speicherplatz gemeinsam zu nutzen. [BORLANDMM.DLL](#) muss mit Anwendungen und DLLs weitergegeben werden, die **ShareMem** einbinden. Wenn eine Anwendung oder DLL **ShareMem** verwendet, ersetzt [BORLANDMM.DLL](#) den Speichermanager dieser Anwendung oder DLL.

Siehe auch

Bibliotheken und Packages (↗ siehe Seite 997)

Packages (↗ siehe Seite 1002)

4.1.1.8.3 Packages

Die folgenden Abschnitte enthalten Informationen zur Erstellung und Compilierung von Packages.

- Package-Deklarationen und Quelltextdateien
- Packages benennen
- Die `requires`-Klausel
- Zirkuläre Bezüge bei Packages vermeiden
- Doppelte Package-Referenzen
- Die `contains`-Klausel
- Redundante Verwendung von Quelltext vermeiden
- Packages compilieren
- Generierte Dateien
- Spezielle Compiler-Direktiven für Packages
- Package-spezifische Befehlszeilenoptionen

Allgemeines zu Packages

Ein Package ist eine auf spezielle Weise compilierte Bibliothek, die von Anwendungen oder der IDE (oder beiden) verwendet wird. Packages ermöglichen eine Neuanordnung des Programmcodes ohne Auswirkungen auf den zugrundeliegenden Quelltext. Dieser Vorgang wird auch als *Partitionierung von Anwendungen* bezeichnet.

Laufzeit-Packages stellen die Funktionalität bereit, die dem Benutzer die Ausführung einer Anwendung ermöglicht. Entwurfszeit-Packages dienen der Installation von Komponenten in der IDE und der Erstellung von speziellen Eigenschaftseditoren für benutzerdefinierte Komponenten. Jedes Package kann sowohl zur Entwurfszeit als auch zur Laufzeit verwendet werden. Entwurfszeit-Packages referenzieren häufig Laufzeit-Packages in ihren `requires`-Klauseln.

In Win32 haben Package-Dateien die Namenserweiterung [.bpl](#) (Borland Package Library). Auf der .NET-Plattform sind Packages .NET-Assemblierungen mit der Erweiterung [.dll](#).

Normalerweise werden Packages beim Start einer Anwendung statisch geladen. Mithilfe der Routinen `LoadPackage` und `UnloadPackage` (in der Unit `SysUtils`) können Sie Packages aber auch dynamisch laden.

Anmerkung: Wenn eine Anwendung Packages verwendet, muss der Name jeder dort eingebundenen Unit weiterhin in der **uses**-Klausel jeder Quelltextdatei angegeben werden, von der die Unit referenziert wird.

Package-Deklarationen und Quelltextdateien

Um eine Verwechslung mit anderen Dateien zu vermeiden, die Delphi-Code enthalten, wird jedes Package in einer separaten Quelltextdatei mit der Namenserweiterung **.dpk** gespeichert. Eine Package-Quelltextdatei enthält keinerlei Typ-, Daten-, Prozedur- oder Funktionsdeklarationen. Sie hat stattdessen folgenden Inhalt:

- Einen Namen für das Package.
- Eine Liste weiterer Packages, die vom neuen Package benötigt werden. Das neue Package wird zu diesen Packages gelinkt.
- Eine Liste der Unit-Dateien, die das compilierte Package enthält. Das Package stellt im Grund eine Hülle für diese Quelltext-Units dar, die die Funktionalität des compilierten Package bereitstellen.

Eine Package-Deklaration hat folgende Form:

```
package PackageName;
  requiresKlausel;
  containsKlausel;
end.
```

PackageName ist ein gültiger Bezeichner. Die Angabe von *requiresKlausel* und *containsKlausel* ist optional. Im folgenden Beispiel wird das Package **DATAAX** deklariert:

```
package DATAAX;
  requires
    rtl,
  contains Db, DBLocal, DBXpress, ... ;
end.
```

Die **requires**-Klausel enthält weitere externe Packages, die vom deklarierten Package verwendet werden. Die Klausel setzt sich aus der Direktive **requires**, einer Liste mit Package-Namen, die durch Kommas voneinander getrennt sind, und einem Strichpunkt zusammen. Wenn ein Package keine weiteren Packages referenziert, ist keine **requires**-Klausel erforderlich.

In der **contains**-Klausel sind die Unit-Dateien aufgeführt, die compiliert und in das Package eingebunden werden. Die Klausel setzt sich aus der Direktive **contains**, einer Liste mit Unit-Namen, die durch Kommas voneinander getrennt sind, und einem Strichpunkt zusammen. Auf einen Unit-Namen können das reservierte Wort **in** und der Name einer Quelltextdatei mit oder ohne Pfadangabe in halben Anführungszeichen folgen. Verzeichnispfade können absolut oder relativ sein. Zum Beispiel:

```
contains MyUnit in 'C:\MyProject\MyUnit.pas';
```

Anmerkung: Auf Thread-Variablen (Variablen, die mit **threadvar** deklariert wurden) in einer Package-Unit können Clients, die dieses Package verwenden, nicht zugreifen.

Packages benennen

Beim Compilieren eines Package werden mehrere Dateien erzeugt. Die Quelldatei für das Package **DATAAX** ist beispielsweise **DATAAX.DPK**. Der Compiler erzeugt daraus eine ausführbare Datei und ein binäres Abbild mit folgenden Namen:

DATAAX.BPL (Win32) bzw. **DATAAX.DLL** (.NET) und **DATAAX.DCP** (Win32) bzw. **DATAAX.DCPIL** (.NET)

Über den Namen **DATAAX** wird das Package in der **requires**-Klausel anderer Packages referenziert oder in einer Anwendung verwendet. Package-Namen müssen innerhalb eines Projekts eindeutig sein.

Die **requires**-Klausel

Die **requires**-Klausel enthält andere externe Packages, die vom aktuellen Package verwendet werden. Diese Klausel funktioniert wie die **uses**-Klausel in einer Unit. Ein externes Package, das in der **requires**-Klausel enthalten ist, wird beim Compilieren automatisch zu jeder Anwendung gelinkt, die sowohl das neue Package als auch eine der Units verwendet, die im externen Package enthalten sind.

Wenn die Unit-Dateien eines Package andere Units referenzieren, die ebenfalls in Packages enthalten sind, sollten diese anderen Packages im **requires**-Abschnitt des ersten Package angegeben werden. Andernfalls lädt der Compiler die referenzierten Units aus den entsprechenden **.dcu**- oder **.dcuил**-Dateien.

4 Zirkuläre Bezüge bei Packages vermeiden

Packages dürfen in ihren **requires**-Klauseln keine zirkulären Bezüge herstellen. Daher müssen folgende Bedingungen erfüllt sein:

- Ein Package darf nicht in seiner eigenen **requires**-Klausel aufgeführt sein.
- Eine Folge von Referenzen darf keine rückwärtsgerichtete Referenz enthalten. Wenn Package A in seinem **requires**-Abschnitt Package B referenziert, kann Package B nicht Package A referenzieren. Wenn Package A Package B und Package B Package C referenziert, kann C nicht A referenzieren.

Doppelte Package-Referenzen

Der Compiler ignoriert doppelte Referenzen in der **requires**-Klausel eines Package. Wegen der besseren Lesbarkeit Ihrer Programme sollten Sie derartige Referenzen aber entfernen.

Die **contains**-Klausel

In der **contains**-Klausel wird festgelegt, welche Unit-Dateien in das Package eingebunden werden sollen. Die **contains**-Klausel darf keine Dateinamenserweiterungen enthalten.

Redundante Verwendung von Quelltext vermeiden

Ein Package kann nicht in der **contains**-Klausel eines anderen Package oder in der **uses**-Klausel einer Unit enthalten sein.

Alle Units, die direkt in der **contains**-Klausel eines Package oder indirekt in **uses**-Klauseln der betreffenden Units enthalten sind, werden zur Compilierzeit in das Package eingebunden. Die Units, die (direkt oder indirekt) in einem Package vorhanden sind, dürfen in keinem der anderen Packages enthalten sein, die in der **requires**-Klausel dieses Package referenziert werden.

Eine Unit kann weder direkt noch indirekt in mehr als einem Package einer Anwendung enthalten sein.

Packages compilieren

Packages werden normalerweise in der IDE unter Verwendung von **.dpk**-Dateien erstellt, die mit dem Projekt-Manager generiert werden. Sie können **.dpk**-Dateien auch direkt in der Befehlszeile anlegen. Bei der Neucompilierung eines Projekts, das ein Package enthält, wird dieses implizit neu compiliert, wenn dies erforderlich ist.

Generierte Dateien

Die folgende Tabelle enthält die Dateien, die beim erfolgreichen Compilieren eines Package generiert werden.

Dateien eines compilierten Package

Namenserweiterung	Inhalt
DCP (Win32) bzw. DCPIL (.NET)	Ein binäres Abbild, das einen Package-Header und die Verkettung aller .dcu -Dateien (Win32) oder .dcu1 -Dateien (.NET) des Package enthält. Für jedes Package wird eine .dcp - oder .dcpil -Datei erzeugt. Der Basisname der Datei entspricht dem Basisnamen der .dpk -Quelltextdatei.
BPL (Win32) bzw. DLL (.NET)	Das Laufzeit-Package. In Win32 ist diese Datei eine DLL mit Borland-spezifischen Eigenschaften. Der Basisname des Package entspricht dem Basisnamen der .dpk -Quelltextdatei.

Package-spezifische Compiler-Direktiven

Die folgende Tabelle enthält die Package-spezifischen Compiler-Direktiven, die in den Quelltext eingefügt werden können.

Spezielle Compiler-Direktiven für Packages

Direktive	Beschreibung
<code>{\$IMPLICITBUILD OFF}</code>	Verhindert, dass ein Package in Zukunft implizit neu compiliert wird. Diese Direktive wird in .dpk -Dateien verwendet, wenn Packages mit Low-Level-Funktionen compiliert werden, die sich nur selten ändern, oder wenn der Quelltext des Package nicht weitergegeben wird.
<code>{\$G-}</code> oder <code>{\$IMPORTEDDATA OFF}</code>	Deaktiviert die Erstellung von Referenzen auf importierte Daten. Diese Direktive führt zu schnelleren Speicherzugriffen, verhindert aber, dass die Unit, in der sie verwendet wird, Variablen in anderen Packages referenzieren kann.
<code>{\$WEAKPACKAGEUNIT ON}</code>	Die Unit wird schwach eingebunden.
<code>{\$DENYPACKAGEUNIT ON}</code>	Verhindert, dass die Unit in ein Package eingebunden wird.
<code>{\$DESIGNONLY ON}</code>	Das Package wird für die Installation in der IDE compiliert (nur in .dpk -Dateien).
<code>{\$RUNONLY ON}</code>	Das Package wird nur als Laufzeit-Package compiliert (nur in .dpk -Dateien).

Die Verwendung von `{$DENYPACKAGEUNIT ON}` im Quelltext verhindert, dass die Unit in ein Package eingebunden wird. Die Angabe von `{$G-}` oder `{$IMPORTEDDATA OFF}` kann bewirken, dass das Package nicht zusammen mit anderen Packages in derselben Anwendung verwendet werden kann.

Bei Bedarf können auch andere Compiler-Direktiven in den Quelltext eines Package aufgenommen werden.

Package-spezifische Befehlszeilenoptionen

Die folgenden Package-spezifischen Optionen stehen für den Befehlszeilen-Compiler zur Verfügung.

Package-spezifische Befehlszeilenoptionen

Option	Beschreibung
<code>-\$G-</code>	Deaktiviert die Erstellung von Referenzen auf importierte Daten. Diese Option führt zu beschleunigten Speicherzugriffen, verhindert aber, dass mit dieser Option compilierte Packages Variablen in anderen Packages referenzieren.
<code>LE Pfad</code>	Gibt das Verzeichnis an, in dem die compilierte Package-Datei abgelegt wird.
<code>LN Pfad</code>	Gibt das Verzeichnis an, in dem die .dcp - oder .dcpil -Datei des Package abgelegt wird.
<code>LUPackageName [;PackageName2;...]</code>	Gibt zusätzliche Laufzeit-Packages für die Verwendung in einer Anwendung an. Wird bei der Compilierung eines Projekts verwendet.

Z	Verhindert, dass ein Package in Zukunft implizit neu compiliert wird. Diese Option wird verwendet, wenn Packages mit Low-Level-Funktionen compiliert werden, die sich nur selten ändern, oder wenn der Quelltext des Package nicht weitergegeben wird.
---	--

Die Verwendung der Option `-$G-` kann zur Folge haben, dass ein Package nicht mehr zusammen mit anderen Packages in derselben Anwendung verwendet werden kann.

Bei Bedarf können für die Compilierung von Packages auch andere Befehlszeilenoptionen angegeben werden.

Anmerkung: Wenn die Option `-LU` auf der .NET-Plattform verwendet wird, kann das Package mit oder ohne `.dll`-Namenserweiterung referenziert werden. Ist die Namenserweiterung `.dll` nicht vorhanden, sucht der Compiler das Package im Suchpfad der Unit und im Suchpfad des Package. Enthält die Angabe für das Package jedoch einen Laufwerkbuchstaben oder das Pfadtrennzeichen, geht der Compiler von einem vollständigen Namen für das Package aus (einschließlich `.dll`-Erweiterung). Wenn Sie in diesem Fall einen vollständigen oder relativen Pfad angeben, die Erweiterung `.dll` aber weglassen, findet der Compiler das Package nicht.

Siehe auch

[Bibliotheken und Packages](#) (siehe Seite 997)

[Dynamisch ladbare Bibliotheken schreiben](#) (siehe Seite 998)

4.1.1.9 Objekt-Interfaces

In diesem Abschnitt wird die Verwendung von Interfaces in Delphi beschrieben.

4.1.1.9.1 Objekt-Interfaces

Ein Objekt-Interface (oder einfach nur Interface) definiert Methoden, die von einer Klasse implementiert werden können. Interfaces werden wie Klassen deklariert. Sie können aber nicht direkt instantiiert werden und verfügen auch nicht über eigene Methodendefinitionen. Es liegt vielmehr in der Verantwortung der Klasse, von der ein Interface unterstützt wird, für die Implementierung von Interface-Methoden zu sorgen. Eine Variable vom Typ eines Interface kann ein Objekt referenzieren, dessen Klasse das betreffende Interface implementiert. Über diese Variable können aber nur die Methoden aufgerufen werden, die in dem Interface deklariert sind.

Interfaces bieten einige Vorteile der Mehrfachvererbung, umgehen aber deren semantische Probleme. Außerdem sind sie bei der Verwendung von verteilten Objektmodellen von größter Bedeutung (z. B. SOAP). Benutzerdefinierte Objekte, die Interfaces unterstützen, können dabei mit Objekten interagieren, die mit C++, Java oder anderen Programmiersprachen entwickelt wurden.

Interface-Typen

Interfaces können wie Klassen nur im äußersten Gültigkeitsbereich eines Programms oder einer Unit, nicht aber in einer Prozedur oder Funktion deklariert werden. Die Deklaration eines Interface-Typs sieht folgendermaßen aus:

```
type InterfaceName = interface (VorfahrInterface)
  ['{GUID}']
  Elementliste
end;
```

(`VorfahrInterface`) und `['{GUID}']` sind für .NET-Interfaces optional.

Warnung: Das Vorfahr-Interface und die GUID-Spezifikation sind für .NET-Interfaces optional, aber für die Unterstützung der Win32 COM-Interoperabilität erforderlich. Wenn auf Ihre Interface über COM zugegriffen werden soll, müssen Sie ein Vorfahr-Interface und die GUID festlegen.

Eine Interface-Deklaration ähnelt in weiten Teilen einer Klassendeklaration. Es gelten jedoch folgende Einschränkungen:

- Die *Elementliste* darf nur Methoden und Eigenschaften enthalten. Felder sind in Interfaces nicht erlaubt.
- Da für Interfaces keine Felder verfügbar sind, müssen die Zugriffsattribute für Eigenschaften (**read** und **write**) Methoden sein.
- Alle Elemente eines Interface sind als **public** deklariert. Sichtbarkeitsattribute und Speicherattribute sind nicht erlaubt. Es kann aber eine Array-Eigenschaft mit der Direktive **default** als Standardeigenschaft deklariert werden.
- Interfaces haben keine Konstruktoren oder Destruktoren. Sie können nicht instantiiert werden, ausgenommen durch Klassen, über die die Methoden des Interface implementiert werden.
- Methoden können nicht als **virtual**, **dynamic**, **abstract** oder **override** deklariert werden. Da Interfaces keine eigenen Methoden implementieren, haben diese Bezeichnungen keine Bedeutung.

Hier ein Beispiel für eine Interface-Deklaration:

```
type
IMalloc = interface(IInterface)
  ['{00000002-0000-0000-C000-000000000046}']
  function Alloc(Size: Integer): Pointer; stdcall;
  function Realloc(P: Pointer; Size: Integer): Pointer; stdcall;
  procedure Free(P: Pointer); stdcall;
  function GetSize(P: Pointer): Integer; stdcall;
  function DidAlloc(P: Pointer): Integer; stdcall;
  procedure HeapMinimize; stdcall;
end;
```

In manchen Interface-Deklarationen wird das reservierte Wort **interface** durch **dispinterface** ersetzt. Dies sind dann (wie auch die Direktiven **dispid**, **readonly** und **writeonly**) plattformspezifische Konstruktionen, die in der Linux-Programmierung nicht verwendet werden.

Interface und Vererbung

Ein Interface erbt wie eine Klasse alle Methoden seines Vorfahren. Interfaces implementieren aber im Gegensatz zu Klassen keine Methoden. Ein Interface erbt die Verpflichtung zur Implementation von Methoden. Diese Verpflichtung geht auf alle Klassen über, die das Interface unterstützen.

In der Deklaration eines Interface kann ein Vorfahr-Interface angegeben werden. Wird kein Vorfahr festgelegt, ist das Interface ein direkter Nachkomme von **IInterface**, das in der Unit **System** definiert ist und den absoluten Vorfahr aller Interfaces darstellt. In Win32 deklariert **IInterface** drei Methoden: **QueryInterface**, **_AddRef** und **_Release**. Auf der .NET-Plattform sind diese Methoden nicht verfügbar und müssen nicht implementiert werden.

Anmerkung: **IInterface** entspricht dem Interface **IUnknown**. Bei der Programmierung plattformunabhängiger Anwendungen sollten Sie jedoch immer **IInterface** verwenden. Das Interface **IUnknown** sollte nur in Win32-Programmen zum Einsatz kommen.

QueryInterface stellt die Mittel bereit, mit deren Hilfe die verschiedenen Interfaces, die ein Objekt unterstützen, angesprochen werden können. **_AddRef** und **_Release** sorgen während der Lebensdauer eines Interface für die Verwaltung der Interface-Referenzen. Die einfachste Art, diese Methoden zu implementieren, besteht darin, die implementierende Klasse von **TIInterfacedObject** in der Unit **System** abzuleiten. Es ist auch möglich, auf diese Methoden zu verzichten und sie als leere Funktionen zu implementieren. COM-Objekte müssen jedoch immer mit **_AddRef** und **_Release** verwaltet werden.

Warnung: **QueryInterface**, **_AddRef** und **_Release** sind für .NET-Interfaces optional, aber zur Unterstützung der Win32 COM-Interoperabilität erforderlich. Wenn auf Ihr Interface über COM zugegriffen werden soll, müssen Sie diese Methoden implementieren.

Identifikation eines Interface

Die Deklaration eines Interface kann einen global eindeutigen Bezeichner (GUID) enthalten, der als String-Literal in eckigen Klammern unmittelbar vor der Elementliste steht. Der GUID-Abschnitt der Deklaration hat folgende Form:

`['{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}']`

Dabei steht jedes *x* für eine hexadezimale Ziffer (0 bis 9 und A bis F). Der Typbibliothekseditor generiert automatisch GUIDs für neue Interfaces. Sie können GUIDs aber auch explizit erstellen, indem Sie im Quelltexteditor die Tastenkombination Strg+Umschalt+G drücken.

Eine GUID (Interface-ID) ist ein binärer 16-Byte-Wert, der ein Interface eindeutig bezeichnet. Wenn ein Interface eine GUID hat, können Sie über eine Interface-Abfrage Referenzen auf seine Implementierungen abrufen.

Anmerkung: Im .NET Framework sind für Interfaces keine GUIDs erforderlich. Sie dienen lediglich der COM-Interoperabilität.

Die Typen `TGUID` und `PGUID`, die in der Unit `System` deklariert sind, werden zur Bearbeitung von GUIDs eingesetzt:

```
4
type
  PGUID = ^TGUID;
  TGUID = packed record
    D1: Longword;
    D2: Word;
    D3: Word;
    D4: array[0..7] of Byte;
  end;
```

Auf der .NET-Plattform können Sie ein Interface wie oben beschrieben kennzeichnen (nach der Interface-Deklaration). In der traditionellen Delphi-Syntax wird das erste Konstrukt in eckigen Klammern nach der Interface-Deklaration jedoch als GUID-Bezeichner und nicht als .NET-Attribut angesehen. (.NET-Attribute beziehen sich immer auf das *nächste* und nicht auf das vorhergehende Symbol.) Sie können einem Interface auch mithilfe des benutzerdefinierten .NET-Attributs `Guid` eine GUID zuordnen. In diesem Fall würden Sie die .NET-spezifische Syntax anwenden und das Attribut unmittelbar vor der Interface-Deklaration einfügen.

Wenn Sie eine typisierte Konstante vom Typ `TGUID` deklarieren, können Sie ihren Wert als String-Literal angeben. Zum Beispiel:

```
const IID_IMalloc: TGUID = '{00000002-0000-0000-C000-00000000046}';
```

In Prozedur- und Funktionsaufrufen kann entweder eine GUID oder ein Interface-Bezeichner als Wert- oder Konstantenparameter vom Typ `TGUID` fungieren. Betrachten Sie beispielsweise die folgende Deklaration:

```
function Supports(Unknown: IInterface; const IID: TGUID): Boolean;
```

`Supports` kann auf zwei Arten aufgerufen werden:

```
if Supports(Allocator, IMalloc) then ...
```

oder

```
if Supports(Allocator, IID_IMalloc) then ...
```

Aufrufkonventionen für Interfaces

Die Standard-Aufrufkonvention für Interface-Methoden ist **register**. Bei Interfaces, die von verschiedenen Modulen gemeinsam benutzt werden, sollten alle Methoden mit **stdcall** deklariert werden. Dies gilt insbesondere dann, wenn diese Module in verschiedenen Programmiersprachen erstellt wurden. In der Win32-Programmierung können mit **safecall** die Methoden von dualen Interfaces implementiert werden.

Interface-Eigenschaften

Auf Eigenschaften, die in einem Interface deklariert werden, kann nur über Ausdrücke vom Typ der Interface zugegriffen werden. Der Zugriff über Variablen vom Typ der Klasse ist nicht möglich. Außerdem sind Interface-Eigenschaften nur in Programmen sichtbar, in denen das Interface compiliert wird.

Da in Interfaces keine Felder verfügbar sind, müssen die Zugriffsattribute für Eigenschaften (**read** und **write**) Methoden sein.

Vorwärtsdeklarationen

Eine Interface-Deklaration, die mit dem reservierten Wort **interface** und einem Strichpunkt endet und keinen Vorfahr, keine GUID und keine Elementliste enthält, wird als Vorwärtsdeklaration bezeichnet. Eine Vorwärtsdeklaration muss durch eine definierende Deklaration desselben Interface innerhalb desselben Typdeklarationsabschnitts aufgelöst werden. Das bedeutet,

dass zwischen einer Vorwärtsdeklaration und ihrer definierenden Deklaration ausschließlich andere Typdeklarationen stehen dürfen.

Vorwärtsdeklarationen ermöglichen voneinander abhängige Interfaces. Zum Beispiel:

```
type
  IControl = interface
  IWindow = interface
  ['{00000115-0000-0000-C000-00000000044}']
  function GetControl(Index: Integer): IControl;
  .

  .
  end;
  IControl = interface
  ['{00000115-0000-0000-C000-00000000049}']
  function GetWindow: IWindow;
  .

  .
  end;
```

Es ist nicht möglich, Interfaces gegenseitig voneinander abzuleiten. So kann beispielsweise nicht `IWindow` von `IControl` und `IControl` von `IWindow` abgeleitet werden.

Siehe auch

[Interfaces implementieren](#) (siehe Seite 1009)

[Interface-Referenzen](#) (siehe Seite 1013)

[Automatisierungsobjekte](#) (siehe Seite 1015)

4.1.1.9.2 Schnittstellen implementieren

Nach der Deklaration muss die Schnittstelle in einer Klasse implementiert werden, bevor sie verwendet werden kann. Die in einer Klasse implementierten Schnittstellen werden in der Klassendeklaration nach dem Namen der Vorfahrklasse angegeben.

Klassendeklarationen

Die Deklaration sieht folgendermaßen aus:

```
type Klassenname = class (Vorfahrklasse, Schnittstelle1, ..., SchnittstelleN)
  Elementliste
end;
```

Zum Beispiel:

```
type
  TMemoryManager = class(TInterfacedObject, IMalloc, IErrorInfo)
  .
  .
  end;
```

Hier wird eine Klasse namens `TMemoryManager` deklariert, die die Schnittstellen `IMalloc` und `IErrorInfo` implementiert. Wenn eine Klasse eine Schnittstelle implementiert, muss sie alle in der Schnittstelle deklarierten Methoden implementieren (oder eine Implementation jeder Methode erben).

Nachstehend sehen Sie die Win32-Deklaration von `TInterfacedObject` in der Unit `System`. Auf der .NET-Plattform ist `TInterfacedObject` ein Alias für `TObject`.

```
type
```

```

TInterfacedObject = class(TObject, IInterface)
protected
  FRefCount: Integer;
  function QueryInterface(const IID: TGUID; out Obj): HResult; stdcall;
  function _AddRef: Integer; stdcall;
  function _Release: Integer; stdcall;
public
  procedure AfterConstruction; override;
  procedure BeforeDestruction; override;
  class function NewInstance: TObject; override;
  property RefCount: Integer read FRefCount;
end;

```

TInterfacedObject implementiert die Schnittstelle IInterface. Daher deklariert und implementiert TInterfacedObject alle drei Methoden von IInterface.

Klassen, die Schnittstellen implementieren, können auch als Basisklassen verwendet werden. (Im ersten der obigen Beispiele wird TMemoryManager als direkter Nachkomme von TInterfacedObject deklariert.) Da auf der Win32-Plattform jede Schnittstelle die Methoden von IInterface erbt, muss eine Klasse, die Schnittstellen implementiert, auch die Methoden QueryInterface, _AddRef und _Release implementieren. TInterfacedObject in der Unit System implementiert diese Methoden und eignet sich aus diesem Grund als Basis für weitere Klassen, die Schnittstellen implementieren. Auf der .NET-Plattform deklariert IInterface diese Methoden nicht, so dass Sie diese Methoden nicht implementieren müssen.

Nach der Implementierung einer Schnittstelle wird jede ihrer Methoden einer Methode der implementierenden Klasse zugeordnet, die denselben Ergebnistyp, dieselbe Aufrufkonvention und dieselbe Anzahl von Parametern hat (wobei entsprechende Parameter auch identische Typen haben müssen). Standardmäßig wird jede Methode der Schnittstelle der gleichnamigen Methode der implementierenden Klasse zugewiesen.

Methodenzuordnungsklausel

Das Standardverfahren der Methodenzuordnung in einer Klassendeklaration kann mithilfe von Methodenzuordnungsklauseln außer Kraft gesetzt werden. Wenn eine Klasse zwei oder mehr Schnittstellen mit identischen Methoden implementiert, können Sie die Namenskonflikte, die sich daraus ergeben, mithilfe von Methodenzuordnungsklauseln lösen.

Eine Methodenzuordnungsklausel sieht folgendermaßen aus:

```
procedure Schnittstelle.SchnittstellenMethode = implementierendeMethode;
```

oder

```
function Schnittstelle.SchnittstellenMethode = implementierendeMethode;
```

Dabei ist *implementierendeMethode* eine in der Klasse deklarierte Methode oder eine Methode eines der Klassenvorfahren. Es kann sich dabei um eine an späterer Stelle deklarierte Methode handeln, nicht aber um eine als **private** deklarierte Methode einer Vorfahrklasse, die in einem anderen Modul deklariert ist.

Ein Beispiel:

```

type
  TMemoryManager = class(TInterfacedObject, IMalloc, IErrorInfo)
    function IMalloc.Alloc = Allocate;
    procedure IMalloc.Free = Deallocate;
  .
  .
  .
end;

```

In dieser Klassendeklaration werden die Methoden Alloc und Free von IMalloc den Methoden Allocate und Deallocate von TMemoryManager zugeordnet.

Die in einer Vorfahrklasse festgelegten Zuordnungen können in Methodenzuordnungsklauseln nicht geändert werden.

Ändern von vererbten Implementierungen

Durch abgeleitete Klassen kann die Art und Weise, wie ein bestimmtes Interface implementiert ist, geändert werden, indem die

Implementierungsmethode überschrieben wird. Dies setzt entweder eine virtuelle oder eine dynamische Implementierungsmethode voraus.

Eine Klasse kann auch eine vollständige Schnittstelle erneut implementieren, die sie von einer Vorfahrklasse geerbt hat. Dies erfordert eine erneute Auflistung der Schnittstelle in der Deklaration der abgeleiteten Klasse. Zum Beispiel:

```
type
  IWindow = interface
  ['{00000115-0000-0000-C000-00000000146}']
  procedure Draw;
  .
  .
  .
end;
TWindow = class(TInterfacedObject, IWindow)// TWindow implementiert IWindow
procedure Draw;
  .
  .
  .
end;
TFrameWindow = class(TWindow, IWindow)// TFrameWindow implementiert IWindow neu
procedure Draw;
  .
  .
  .
end;
```

Durch eine erneute Implementierung einer Schnittstelle wird die geerbte Implementierung derselben Schnittstelle verdeckt. Aus diesem Grund haben Methodenzuordnungsklauseln in der Vorgängerklasse keine Auswirkung auf die erneut implementierte Schnittstelle.

Implementieren von Schnittstellen durch Delegation (nur Win32)

Auf der Win32-Plattform ermöglicht es die Direktive **implements**, die Implementierung einer Schnittstelle an eine Eigenschaft der implementierenden Klasse zu delegieren. Zum Beispiel:

```
property MyInterface: IMyInterface read FMyInterface implements IMyInterface;
```

Hier wird eine Eigenschaft namens `MyInterface` deklariert, die die Schnittstelle `IMyInterface` implementiert.

Die Direktive **implements** muss der letzte Bezeichner in der Eigenschaftsdeklaration sein und kann mehrere Schnittstellen enthalten, die durch Kommas voneinander getrennt sind. Die "beauftragte" Eigenschaft muss folgende Bedingungen erfüllen:

- Sie muss vom Typ der Klasse oder der Schnittstelle sein.
- Sie darf keine Array-Eigenschaft sein und keinen Index-Bezeichner verwenden.
- Sie muss über einen **read**-Bezeichner verfügen. Wenn es für die Eigenschaft eine **read**-Methode gibt, muss diese die standardmäßige Aufrufkonvention **register** verwenden. Außerdem darf die Methode nicht dynamisch (wohl aber virtuell) sein oder die Direktive **message** verwenden.

Die Klasse, die zur Implementierung der delegierten Schnittstelle verwendet wird, muss von `TAggregatedObject` abgeleitet sein.

Anmerkung: Wegen der Beschränkungen von CLR wird die Direktive **implements** nicht auf der .NET-Plattform unterstützt.

Delegieren an eine Eigenschaft vom Typ einer Schnittstelle (nur Win32)

Wenn die "beauftragte" Eigenschaft den Typ einer Schnittstelle hat, muss diese Schnittstelle bzw. die übergeordnete Schnittstelle in der Vorfahrenliste der Klasse enthalten sein, in der die Eigenschaft deklariert wird. Die beauftragte Eigenschaft muss ein Objekt zurückgeben, dessen Klasse die mit der Direktive **implements** angegebene Schnittstelle vollständig implementiert. Dabei dürfen keine Methodenzuordnungsklauseln verwendet werden. Zum Beispiel:

```

type
  IMyInterface = interface
    procedure P1;
    procedure P2;
  end;
TMyClass = class(TObject, IMyInterface)
  FMyInterface: IMyInterface;
  property MyInterface: IMyInterface read FMyInterface implements IMyInterface;
end;
var
  MyClass : TMyClass;
  MyInterface : IMyInterface;
begin
  MyClass := TMyClass.Create;
  MyClass.FMyInterface := ...//Ein Objekt, dessen Klasse IMyInterface implementiert
  MyInterface := MyClass;
  MyInterface.P1;
end;

```

Delegieren an eine Eigenschaft vom Typ einer Klasse (nur Win32)

Wenn die "beauftragte" Eigenschaft den Typ einer Klasse hat, werden diese Klasse und ihre Vorfahren nach Methoden durchsucht, welche das angegebene Interface implementieren. Danach werden bei Bedarf die umgebende Klasse und ihre Vorfahren durchsucht. Es ist daher möglich, einige Methoden in der durch die Eigenschaft bezeichneten Klasse zu deklarieren, andere dagegen in der Klasse, in der die Eigenschaft deklariert ist. Methodenzuordnungsklauseln können wie üblich verwendet werden, um Mehrdeutigkeiten aufzulösen oder um eine bestimmte Methode anzugeben. Ein Interface kann immer nur durch eine einzige Eigenschaft mit dem Typ einer Klasse implementiert werden. Zum Beispiel:

```

type
  IMyInterface = interface
    procedure P1;
    procedure P2;
  end;
TMyImplClass = class
  procedure P1;
  procedure P2;
end;
TMyClass = class(TInterfacedObject, IMyInterface)
  FMyImplClass: TMyImplClass;
  property MyImplClass: TMyImplClass read FMyImplClass implements IMyInterface;
  procedure IMyInterface.P1 = MyP1;
  procedure MyP1;
end;
procedure TMyImplClass.P1;
.
.
.
procedure TMyImplClass.P2;
.
.
.
procedure TMyClass.MyP1;
.
.
.
var
  MyClass : TMyClass;
  MyInterface : IMyInterface;
begin
  MyClass := TMyClass.Create;
  MyClass.FMyImplClass := TMyImplClass.Create;
  MyInterface := MyClass;
  MyInterface.P1;    // Aufruf von TMyClass.MyP1;

```

```
MyInterface.P2;      // Aufruf von TImplClass.P2;
end;
```

Siehe auch

Objektschnittstellen ([siehe Seite 1006](#))

Schnittstellenreferenzen ([siehe Seite 1013](#))

Automatisierungsobjekte ([siehe Seite 1015](#))

4.1.1.9.3 Schnittstellenreferenzen

Eine mit einem Schnittstellentyp deklarierte Variable kann Instanzen jeder Klasse referenzieren, die von der Schnittstelle implementiert wird. In den folgenden Abschnitten werden Schnittstellenreferenzen und verwandte Themen beschrieben.

Schnittstellenreferenzen implementieren

Mithilfe von Schnittstellenreferenzen können Schnittstellenmethoden auch aufgerufen werden, wenn zur Compilierzeit noch nicht bekannt ist, wo die Schnittstelle implementiert wird. Es gelten jedoch folgende Einschränkungen:

- Mit Ausdrücken vom Typ einer Schnittstelle kann nur auf Methoden und Eigenschaften zugegriffen werden, die in der Schnittstelle deklariert sind. Ein Zugriff auf andere Elemente der implementierenden Klasse ist nicht möglich.
- Ein Ausdruck vom Typ einer Schnittstelle kann nur dann ein Objekt referenzieren, dessen Klasse ein abgeleitete Schnittstelle implementiert, wenn die Klasse (bzw. eine übergeordnete Klasse) die abgeleitete Schnittstelle explizit implementiert.

Zum Beispiel:

```
type
  IAncestor = interface
end;
IDescendant = interface(IAncestor)
  procedure P1;
end;
TSomething = class(TInterfacedObject, IDescendant)
  procedure P1;
  procedure P2;
end;
.
.
.
var
  D: IDescendant;
  A: IAncestor;
begin
  D := TSomething.Create;  // Funktioniert!
  A := TSomething.Create;  // Fehler
  D.P1; // Funktioniert!
  D.P2; // Fehler
end;
```

In diesem Beispiel ist A als Variable des Typs IAncestor deklariert. Da IAncestor nicht in den von TSomething implementierten Schnittstellen enthalten ist, kann A keine Instanz von TSomething zugewiesen werden. Wenn wir aber die Deklaration von TSomething folgendermaßen ändern:

```
TSomething = class(TInterfacedObject, IAncestor, IDescendant)
.
.
.
```

wird der erste Fehler zu einer gültigen Zuweisung. D ist als Variable des Typs IDescendant deklariert. Solange D eine Instanz von TSomething referenziert, kann damit nicht auf die Methode P2 von TSomething zugegriffen werden (P2 ist keine Methode von IDescendant). Wenn wir aber die Deklaration von D folgendermaßen ändern:

D: TSomething;
wird der zweite Fehler zu einem gültigen Methodenaufruf.

Schnittstellenreferenzen werden auf der Win32-Plattform über einen Referenzähler verwaltet, der auf den von `IInterface` geerbten Methoden `_AddRef` und `_Release` basiert. Diese Methoden sowie die Referenzzählung im Allgemeinen können nicht auf der .NET-Plattform, einer Umgebung mit Speicherbereinigung (Garbage Collection), angewendet werden. Bei Verwendung der Standardimplementierung der Referenzzählung braucht ein Objekt, das ausschließlich durch Schnittstellen referenziert wird, nicht manuell freigegeben zu werden. Seine Freigabe erfolgt automatisch, wenn der Referenzähler den Wert Null erreicht. Einige Klassen implementieren Schnittstellen, um diese Standard-Lebensdauerverwaltung zu umgehen, und einige Hybridobjekte verwenden die Referenzzählung nur, wenn das Objekt keinen Eigentümer hat.

Für globale Variablen vom Typ einer Schnittstelle ist nur der Initialisierungswert **nil** zulässig.

Um festzustellen, ob ein Ausdruck vom Typ einer Schnittstelle ein Objekt referenziert, übergeben Sie ihn an die Standardfunktion **Assigned**.

Zuweisungskompatibilität von Schnittstellen

Variablen eines bestimmten Klassentyps sind zu jedem Schnittstellentyp zuweisungskompatibel, der von der Klasse implementiert wird. Variablen eines Schnittstellentyps sind zu jedem Schnittstellentyp kompatibel, von dem er abgeleitet ist. Jeder Variable vom Typ einer Schnittstelle kann der Wert **nil** zugewiesen werden.

Ein Ausdruck vom Typ einer Schnittstelle kann einer Variante zugewiesen werden. Wenn eine Schnittstelle vom Typ `IDispatch` oder ein Nachkomme von `IDispatch` ist, hat die resultierende Variante den Typencode `varDispatch`. Andernfalls hat die Variante den Typencode `varUnknown`.

Eine Variante mit dem Typencode `varEmpty`, `varUnknown` oder `varDispatch` kann einer `IInterface`-Variablen zugewiesen werden. Varianten mit dem Typencode `varEmpty` und `varDispatch` können an `IDispatch`-Variablen zugewiesen werden.

Schnittstellen-Umwandlungen

Ein Ausdruck vom Typ einer Schnittstelle kann in den Typ Variant konvertiert werden. Wenn eine Schnittstelle vom Typ `IDispatch` oder ein Nachkomme von `IDispatch` ist, hat die resultierende Variante den Typencode `varDispatch`. Andernfalls hat die Variante den Typencode `varUnknown`.

Eine Variante mit dem Typencode `varEmpty`, `varUnknown` oder `varDispatch` kann in `IInterface` umgewandelt werden. Varianten mit dem Typencode `varEmpty` oder `varDispatch` können in den Typ `IDispatch` konvertiert werden.

Schnittstellen-Abfragen

Mithilfe des Operators `as` können Schnittstellen-Umwandlungen durchgeführt werden. Dieser Vorgang wird als Schnittstellen-Abfrage bezeichnet. Eine Schnittstellen-Abfrage generiert auf der Basis des (zur Laufzeit vorliegenden) Objekttyps aus einer Objektreferenz oder einer anderen Schnittstellenreferenz einen Ausdruck vom Typ einer Schnittstelle. Die Syntax für eine Schnittstellen-Abfrage lautet

Objekt as *Schnittstelle*

Objekt ist entweder ein Ausdruck vom Typ einer Schnittstelle oder einer Variante, oder er bezeichnet eine Instanz einer Klasse, die eine Schnittstelle implementiert. *Schnittstelle* kann jede mit einer GUID deklarierte Schnittstelle sein.

Wenn *Objekt* den Wert **nil** hat, liefert die Schnittstellen-Abfrage als Ergebnis **nil**. Andernfalls wird die GUID der Schnittstelle an die Methode `QueryInterface` von *Objekt* übergeben. Wenn `QueryInterface` einen Wert ungleich Null zurückgibt, wird eine Exception ausgelöst. Ist der Rückgabewert dagegen Null (d. h. die Schnittstelle ist in der Klasse von *Objekt* implementiert), ergibt die Schnittstellen-Abfrage eine Referenz auf *Objekt*.

Siehe auch

Objektschnittstellen (siehe Seite 1006)

Schnittstellen implementieren (siehe Seite 1009)

Automatisierungsobjekte (siehe Seite 1015)

4.1.1.9.4 Automatisierungsobjekte (nur Win32)

Ein Objekt, dessen Klasse das in der Unit `System` deklarierte Interface `IDispatch` implementiert, ist ein Automatisierungsobjekt.

Der Zugriff auf Automatisierungsobjekte erfolgt über Varianten. Wenn ein Automatisierungsobjekt über eine Variante referenziert wird, können die Eigenschaften des Objekts durch einen Aufruf der entsprechenden Objektmethoden über die Variante gelesen und geändert werden. Zu diesem Zweck muss `ComObj` in die `uses`-Klausel einer der Units, des Programms oder der Bibliothek aufgenommen werden.

Typen für Dispatch-Interfaces

Über Dispatch-Interfaces werden die Methoden und Eigenschaften definiert, die ein Automatisierungsobjekt mithilfe des `IDispatch`-Interface implementiert. Aufrufe der Methoden eines Dispatch-Interface werden zur Laufzeit über die Methode `Invoke` von `IDispatch` weitergeleitet. Ein Dispatch-Interface kann nicht von einer Klasse implementiert werden.

Die Syntax zur Deklaration eines Dispatch-Interface lautet:

```
type interfaceName = dispinterface
  ['{GUID}']
  Elementliste
end;
```

Die Angabe von `['{GUID}']` ist optional. *Elementliste* besteht aus Eigenschafts- und Methodendeklarationen. Eine Dispatch-Interface-Deklaration hat große Ähnlichkeit mit der Deklaration eines normalen Interface, mit dem Unterschied, dass kein Vorfahr angegeben werden kann. Zum Beispiel:

```
type
  IStringsDisp = dispinterface
  ['{EE05DFE2-5549-11D0-9EA9-0020AF3D82DA}']
    property ControlDefault[Index: Integer]: OleVariant dispid 0; default;
    function Count: Integer; dispid 1;
    property Item[Index: Integer]: OleVariant dispid 2;
    procedure Remove(Index: Integer); dispid 3;
    procedure Clear; dispid 4;
    function Add(Item: OleVariant): Integer; dispid 5;
    function _NewEnum: IUnknown; dispid -4;
  end;
```

Methoden für Dispatch-Interfaces

Die Methoden für ein Dispatch-Interface stellen Prototypen für Aufrufe der Methode `Invoke` dar, die zur zugrundeliegenden Implementation von `IDispatch` gehört. Um für eine Methode eine Dispatch-ID für die Automatisierung festzulegen, nehmen Sie in die Methodendeklaration die Direktive `dispid` und eine Integer-Konstante auf. Die Verwendung einer bereits vergebenen Dispatch-ID führt zu einem Fehler.

Eine Methode, die in einem Dispatch-Interface deklariert wird, darf neben `dispid` keine weiteren Direktiven enthalten. Parameter und Ergebnistypen müssen automatisierbar sein. Automatisierbare Typen sind: **Byte**, **Currency**, **Real**, **Double**, **Longint**, **Integer**, **Single**, **Smallint**, **AnsiString**, **WideString**, **TDateTime**, **Variant**, **OleVariant** und **WordBool** sowie alle Interface-Typen.

Eigenschaften für Dispatch-Interfaces

Die Eigenschaften eines Dispatch-Interface enthalten keine Zugriffsbezeichner. Sie können entweder mit der Direktive **read only** oder mit der Direktive **write only** deklariert werden. Zur Festlegung einer Dispatch-ID für eine Eigenschaft nehmen Sie die Direktive `dispid` in die Eigenschaftsdeklaration auf. Auf `dispid` muss eine Integer-Konstante folgen. Die Verwendung einer bereits vergebenen Dispatch-ID führt zu einem Fehler. Mithilfe der Direktive `default` können Sie eine Array-Eigenschaft als Standardeigenschaft für das Interface festlegen. Eigenschaftsdeklarationen für Dispatch-Interfaces dürfen keine anderen als die genannten Direktiven enthalten.

Zugriff auf Automatisierungsobjekte

Da Aufrufe von Methoden des Automatisierungsobjekts zur Laufzeit gebunden werden, sind keine Methodendeklarationen erforderlich. Die Gültigkeit solcher Aufrufe wird vom Compiler nicht überprüft.

Das folgende Beispiel demonstriert Aufrufe von Automatisierungsmethoden. Die (in `ComObj` definierte) Funktion `CreateOleObject` gibt eine `IDispatch`-Referenz auf ein Automatisierungsobjekt zurück und ist zuweisungskompatibel zur Variante **Word**.

```
4
var
  Word: Variant;
begin
  Word := CreateOleObject('Word.Basic');
  Word.FileNew('Normal');
  Word.Insert('Das ist die erste Zeile '#13);
  Word.Insert('Das ist die zweite Zeile '#13);
  Word.FileSaveAs('c:\temp\test.txt', 3);
end;
```

Parameter vom Typ eines Interface können an Automatisierungsmethoden übergeben werden.

Die Übergabe binärer Daten zwischen Automatisierungs-Controllern und Automatisierungs-Servern erfolgt bevorzugt über variante Arrays mit dem Elementtyp `varByte`. Da die Daten derartiger Arrays nicht übersetzt werden, können Sie mit den Routinen `VarArrayLock` und `VarArrayUnlock` effizient darauf zugreifen.

Syntax für Aufrufe von Automatisierungsmethoden

Aufrufe von Methoden eines Automatisierungsobjekts und Zugriffe auf die Eigenschaften des Objekts folgen einer ähnlichen Syntax wie normale Methodenaufrufe und Eigenschaftszugriffe. Beim Aufruf einer Automatisierungsmethode können aber sowohl Positionsparameter als auch benannte Parameter verwendet werden (benannte Parameter werden allerdings nicht von allen Automatisierungs-Servern unterstützt).

Ein Positionsparameter ist einfach ein Ausdruck. Ein benannter Parameter setzt sich aus einem Parameterbezeichner, dem Symbol `:=` und einem Ausdruck zusammen. In einem Methodenaufruf müssen Positionsparameter immer vor benannten Parametern angegeben werden. Die Reihenfolge der benannten Parameter ist beliebig.

Bei bestimmten Automatisierungs-Servern können in Methodenaufrufen Parameter weggelassen werden. In diesem Fall werden Standardwerte verwendet. Zum Beispiel:

```
Word.FileSaveAs('test.doc');
Word.FileSaveAs('test.doc', 6);
Word.FileSaveAs('test.doc',,'secret');
Word.FileSaveAs('test.doc', Password := 'secret');
Word.FileSaveAs(Password := 'secret', Name := 'test.doc');
```

Sie können einer Automatisierungsmethode neben Parametern vom Typ **Integer** und **String** auch reelle, Boolesche und variante Typen übergeben. Parameterausdrücke, die nur aus einer Variablenreferenz bestehen, werden als Referenz übergeben. Dies gilt allerdings nur, wenn die Variablenreferenz einen der folgenden Typen hat: **Byte**, **Smallint**, **Integer**, **Single**, **Double**, **Currency**, **TDateTime**, **AnsiString**, **WordBool** oder Variant. Hat der Ausdruck einen anderen Typ oder handelt es sich nicht um eine Variable, wird der Parameter als Wert übergeben. Wird ein Parameter per Referenz an eine Methode übergeben, die einen Wert-Parameter erwartet, ermittelt COM den Wert aus der Referenz. Dagegen führt die Übergabe eines Wert-Parameters an eine Methode, die eine Übergabe per Referenz erwartet, zu einem Fehler.

Duale Schnittstellen

Ein duales Interface ist ein Interface, das sowohl die Bindung zur Compilierzeit als auch die Laufzeitbindung mittels Automatisierung unterstützt. Duale Interfaces müssen Nachkommen von `IDispatch` sein.

Alle Methoden eines dualen Interface (mit Ausnahme der von `IInterface` und `IDispatch` geerbten Methoden) müssen die Aufrufkonvention **safecall** verwenden. Alle Methodenparameter und Ergebnistypen müssen den Anforderungen der Automatisierung entsprechen. (Automatisierbare Typen sind: **Byte**, **Currency**, **Real**, **Double**, **Real48**, **Integer**, **Single**, **Smallint**,

AnsiString, ShortString, TDateTime, Variant, OleVariant und WordBool.

Siehe auch

Objektschnittstellen ([siehe Seite 1006](#))

Schnittstellen implementieren ([siehe Seite 1009](#))

Schnittstellenreferenzen ([siehe Seite 1013](#))

4.1.1.10 Speicherverwaltung auf der Win32-Plattform

In diesem Abschnitt wird die Speicherverwaltung in Delphi-Programmen auf der Win32- und der .NET-Plattform beschrieben.

4.1.1.10.1 Speicherverwaltung auf der Win32-Plattform

Im Folgenden wird beschrieben, wie die Speicherverwaltung auf der Win32-Plattform gehandhabt wird. Außerdem finden Sie hier eine kurze Erläuterung der Speicheranforderungen von Variablen.

Der Speichermanager (nur Win32)

Der Speichermanager ist für alle Operationen zuständig, mit denen eine Anwendung dynamisch Speicher zuweist oder freigibt. Er wird von den Standardprozeduren New, Dispose, GetMem, ReallocMem und FreeMem und bei der Zuweisung von Speicher an alle Objekte und lange Strings verwendet.

Der Speichermanager ist speziell auf Anwendungen zugeschnitten, die sehr viele Blöcke kleiner bis mittlerer Größe belegen. Dies ist typisch für objektorientierte Anwendungen und für Anwendungen, die String-Daten verarbeiten. Der Speichermanager ist für eine effiziente Ausführung (Hochgeschwindigkeit und wenig Speicher-Overhead) in Einzel- und Multi-Thread-Anwendungen optimiert. Andere Speichermanager wie die Implementierungen von GlobalAlloc und LocalAlloc und die Unterstützung des privaten Heap in Windows sind in solchen Situationen weniger geeignet und verlangsamen eine Anwendung spürbar, wenn sie direkt verwendet werden.

Um die bestmögliche Leistung zu erzielen, arbeitet der Speichermanager direkt mit der Win32-API für virtuellen Speicher zusammen (über die Funktionen VirtualAlloc und VirtualFree). Der Speichermanager unterstützt einen Adressraum für den Benutzermodus von bis zu 4 GB.

Die Blöcke des Speichermanagers belegen immer ein Vielfaches von vier Byte und enthalten immer einen vier Byte großen Header, in dem die Größe des Blocks und weitere Statusbits gespeichert sind. The start address of memory blocks are always aligned on at least 8-byte boundaries, or optionally on 16-byte boundaries, which improves performance when addressing them.

Der Speichermanager wendet einen Algorithmus an, der künftige Neuzuweisungen von Blöcken antizipiert. Dadurch wird die Verringerung der Ausführungsgeschwindigkeit aufgehoben, die in der Regel mit solchen Operationen einhergeht. Dieser Zuweisungsalgorithmus verhindert auch die Adressraumfragmentierung.

Der Speichermanager stellt einen Mechanismus für die gemeinsame Nutzung bereit, für den keine externe DLL erforderlich ist.

Der Speichermanager enthält Berichtsfunktionen, die Anwendungen dabei unterstützen, ihre eigene Speicherverwendung und potentielle Speicherlecks zu überwachen.

Der Speichermanager stellt die beiden Prozeduren GetMemoryManagerState und GetMemoryMap bereit, über die Anwendungen Informationen über den Status des Speichermanagers und ein detailliertes Abbild der Speicherverwendung abrufen können.

Variablen

Globale Variablen werden im Datensegment der Anwendung zugewiesen und bleiben bis zur Beendigung des Programms erhalten. Lokale Variablen, die innerhalb von Prozeduren und Funktionen deklariert sind, werden auf dem Stack der Anwendung abgelegt. Wenn eine Prozedur oder Funktion aufgerufen wird, reserviert sie auf dem Stack Speicherplatz für ihre lokalen

Variablen. Bei der Beendigung der Prozedur oder Funktion werden die lokalen Variablen wieder freigegeben. Variablen können aber aufgrund von Optimierungsaktionen des Compilers auch zu einem früheren Zeitpunkt freigegeben werden.

In Win32 wird der Stack einer Anwendung durch zwei Werte definiert: der Mindestgröße und der Maximalgröße. Diese Werte werden über die Compiler-Direktiven `$MINSTACKSIZE` und `$MAXSTACKSIZE` gesteuert. Die Voreinstellung lautet 16.384 (16 KB) bzw. 1.048.576 (1 MB). Eine Anwendung hat an Stack nie weniger als die Mindestgröße und nie mehr als die Maximalgröße zur Verfügung. Wenn beim Start einer Anwendung weniger Speicher zur Verfügung steht, als es der Wert für die Mindestgröße des Stack vorschreibt, gibt Windows eine entsprechende Fehlermeldung aus.

Wenn eine Win32-Anwendung mehr Stack benötigt als die angegebene Mindestgröße, wird ihr in Blöcken von vier KB automatisch weiterer Speicher zugewiesen. Schlägt die Zuweisung fehl, weil nicht mehr Speicher vorhanden ist oder die Maximalgröße des Stack überschritten würde, wird eine **EStackOverflow**-Exception ausgelöst. (Die Stack-Überlaufprüfung wird automatisch durchgeführt. Die Compiler-Direktive `$S` zum Ein- und Ausschalten dieser Prüfung wurde aber aus Gründen der Abwärtskompatibilität beibehalten.)

Der Speicher für dynamische Variablen, die Sie mit den Prozeduren `GetMem` oder `New` erzeugen, wird auf dem Heap reserviert. Die Variablen bleiben bis zu einem entsprechenden `FreeMem`- bzw. `Dispose`-Aufruf erhalten.

Lange Strings, `WideStrings`, dynamische Arrays, Varianten und Interfaces werden auf dem Heap zugewiesen. Ihr Speicher wird aber dennoch automatisch verwaltet.

Siehe auch

[Interne Datenformate](#) (siehe Seite 1018)

[Speicherverwaltung auf der .NET-Plattform](#) (siehe Seite 1027)

[Konfigurieren des Speichermanagers](#) (siehe Seite 95)

[Vergrößern des Adressraums des Speichermanagers über 2 GB hinaus](#) (siehe Seite 97)

[Registrieren von Speicherlecks](#) (siehe Seite 100)

[Überwachen des Speichermanagers](#) (siehe Seite 99)

[Gemeinsame Nutzung von Speicher](#) (siehe Seite 101)

4.1.1.10.2 Interne Datenformate

In den folgenden Themen werden die internen Datenformate von Delphi-Datentypen beschrieben.

Integer-Typen

Das Format, in dem eine Variable eines Integer-Typs dargestellt wird, hängt von ihren Bereichsgrenzen ab.

- Liegen beide Grenzen im Bereich 128..127 (Shortint), wird die Variable als Byte mit Vorzeichen gespeichert.
- Liegen beide Grenzen im Bereich 0..255 (Byte), wird die Variable als Byte ohne Vorzeichen gespeichert.
- Liegen beide Grenzen im Bereich 32768..32767 (Smallint), wird die Variable als Word mit Vorzeichen gespeichert.
- Liegen beide Grenzen im Bereich 0..65535 (Word), wird die Variable als Word ohne Vorzeichen gespeichert.
- Liegen beide Grenzen im Bereich 2147483648..2147483647 (Longint), wird die Variable als Double Word mit Vorzeichen gespeichert.
- Liegen beide Grenzen im Bereich 0..4294967295 (Longword), wird die Variable als Double Word ohne Vorzeichen gespeichert.
- In allen anderen Fällen wird die Variable als Vierfach-Word mit Vorzeichen (Int64) gespeichert.

Anmerkung: Ein "Word" belegt zwei Bytes.

Zeichentypen

Auf der Win32-Plattform wird ein Zeichen vom Typ **Char**, **AnsiChar** oder einem Teilbereich von **Char** als Byte ohne Vorzeichen gespeichert. Eine Variable vom Typ **WideChar** wird als Word ohne Vorzeichen gespeichert.

Auf der .NET-Plattform entspricht **Char** **WideChar**.

Boolesche Typen

Eine Variable vom Typ **Boolean** oder **ByteBool** wird als **Byte**, eine Variable vom Typ **WordBool** als **Word** und eine Variable vom Typ **LongBool** als **Longint** gespeichert.

Eine Variable vom Typ **Boolean** kann die Werte 0 (**False**) und 1 (**True**) annehmen. Variablen vom Typ **ByteBool**, **WordBool** und **LongBool** können die Werte 0 (**False**) oder ungleich 0 (**True**) annehmen.

Aufzählungstypen

Eine Variable eines Aufzählungstyps wird als Byte ohne Vorzeichen gespeichert, wenn die Aufzählung nicht mehr als 256 Werte umfasst und der Typ im Status `{$Z1}` (Voreinstellung) deklariert wurde. Enthält der Aufzählungstyp mehr als 256 Werte oder wurde er im Status `{$Z2}` deklariert, wird die Variable als Word ohne Vorzeichen gespeichert. Wird ein Aufzählungstyp im Status `{$Z4}` deklariert, wird die Variable als Double Word ohne Vorzeichen gespeichert.

Reelle Typen

Reelle Typen speichern die binäre Entsprechung des Vorzeichens (+ oder -), eines Exponenten und einer Mantisse. Ein reeller Wert wird in folgender Form dargestellt:

$+\text{- Mantisse} \cdot 2^{\text{Exponent}}$

Dabei wird für die *Mantisse* links des binären Dezimalpunkts ein einziges Bit verwendet (d. h. $0 \leq \text{Mantisse} < 2$).

In den folgenden Abbildungen befindet sich das höchstwertige Bit immer auf der linken Seite, das niedrigstwertige Bit immer auf der rechten Seite. Die Zahlen am oberen Rand geben die Breite jedes Feldes in Bit an. Die Elemente ganz links werden an den höchsten Adressen gespeichert. Bei einem Real48-Wert wird beispielsweise *e* im ersten Byte, *f* in den nächsten fünf Byte und *s* im höchstwertigen Bit des letzten Byte gespeichert.

Der Typ Real48

Die folgenden Erläuterungen zu **Real48** beziehen sich nur auf die Win32-Plattform. Der Typ **Real48** wird auf der .NET-Plattform nicht unterstützt.

Auf der Win32-Plattform wird eine Real48-Zahl mit sechs Byte (48 Bit) in drei Felder unterteilt:

1	39	8
<i>s</i>	<i>f</i>	<i>e</i>

Wenn gilt $0 < e \leq 255$, ergibt sich der Wert *v* der Zahl folgendermaßen:

$$v = (1)^s \cdot 2^{(e-127)} \cdot (1.f)$$

Wenn *e* = 0, ist *v* = 0.

Der Typ **Real48** eignet sich nicht zum Speichern von NaN-Werten (Not a Number = keine Zahl) und unendlichen Werten. Das Speichern von NaN-Werten und unendlichen Werten in **Real48**-Variablen führt zu einem Überlauffehler.

Der Typ Single

Eine Single-Zahl mit vier Byte (32 Bit) wird in drei Felder unterteilt.

1	8	23
s	e	f

Der Wert v der Zahl ergibt sich folgendermaßen:

Wenn $0 < e < 255$, ist $v = (1)^s \cdot 2^{(e-127)} \cdot (1.f)$

Wenn $e = 0$ und $f <> 0$, ist $v = (1)^s \cdot 2^{(126)} \cdot (0.f)$

Wenn $e = 0$ und $f = 0$, ist $v = (1)^s \cdot 0$

Wenn $e = 255$ und $f = 0$, ist $v = (1)^s \cdot \text{Inf}$

Wenn $e = 255$ und $f <> 0$, ist v ein NaN

Der Typ Double

Eine Double-Zahl mit acht Byte (64 Bit) wird in drei Felder unterteilt.

1	11	52
s	e	f

Der Wert v der Zahl ergibt sich folgendermaßen:

Wenn $0 < e < 2047$, ist $v = (1)^s \cdot 2^{(e-1023)} \cdot (1.f)$

Wenn $e = 0$ und $f <> 0$, ist $v = (1)^s \cdot 2^{(1022)} \cdot (0.f)$

Wenn $e = 0$ und $f = 0$, ist $v = (1)^s \cdot 0$

Wenn $e = 2047$ und $f = 0$, ist $v = (1)^s \cdot \text{Inf}$

Wenn $e = 2047$ und $f <> 0$, ist v ein NaN

Der Typ Extended

Eine Extended-Zahl mit zehn Byte (80 Bit) wird in vier Felder unterteilt:

1	15	1	63
s	e	i	f

Der Wert v der Zahl ergibt sich folgendermaßen:

Wenn $0 \leq e < 32767$, ist $v = (1)^s \cdot 2^{(e-16383)} \cdot (i.f)$

Wenn $e = 32767$ und $f = 0$, ist $v = (1)^s \cdot \text{Inf}$

Wenn $e = 32767$ und $f <> 0$, ist v ein NaN

Anmerkung: Auf der .NET-Plattform wird statt dem Typ Extended der Typ Double verwendet.

Der Typ Comp

Eine Comp-Zahl mit acht Byte (64 Bit) wird als 64-Bit-Integer mit Vorzeichen gespeichert.

Anmerkung: Auf der .NET-Plattform wird statt dem Typ Comp der Typ Int64 verwendet.

Der Typ Currency

Eine Currency-Zahl mit acht Byte (64 Bit) wird als skaliertes 64-Bit-Integer mit Vorzeichen gespeichert. Dabei stehen die vier niedrigstwertigen Ziffern implizit für vier Dezimalstellen.

Zeigertypen

Eine Zeigervariable wird als 32-Bit-Adresse gespeichert und belegt vier Byte. Der Zeigerwert **nil** wird als Null gespeichert.

Anmerkung: Auf der .NET-Plattform variiert die Größe eines Zeigers zur Laufzeit. Deshalb ist `SizeOf(pointer)` keine Konstante zur Compilierzeit wie auf der Win32-Plattform.

Kurze String-Typen

Die Anzahl der Byte eines Strings ergibt sich aus seiner maximalen Länge + 1. Das erste Byte enthält die aktuelle dynamische Länge des Strings, die folgenden Byte seine Zeichen.

Das Längenbyte und die Zeichen werden als Werte ohne Vorzeichen betrachtet. Die maximale Länge eines Strings beträgt 255 Zeichen plus ein Längenbyte (`string[255]`).

Anmerkung: Auf der .NET-Plattform ist der kurze String-Typ als Array von vorzeichenlosen Bytes implementiert.

Lange String-Typen

Eine lange String-Variable belegt vier Byte, die einen Zeiger auf einen dynamisch zugewiesenen String enthalten. Wenn eine lange String-Variable leer ist (also einen String der Länge 0 enthält), ist der String-Zeiger **nil**, und der Variablen wird kein dynamischer Speicher zugewiesen. Andernfalls referenziert der String-Zeiger einen dynamisch zugewiesenen Speicherblock, der neben dem String-Wert eine Längenangabe und einen Referenzzähler von je 32 Bit enthält. Die nachstehende Tabelle zeigt den Aufbau eines Speicherblocks für einen langen String.

Aufbau des Speichers für einen langen String (nur Win32)

Offset	Inhalt
-8	32-Bit-Referenzzähler
-4	Länge in Byte
0..Länge - 1	Zeichen-String
Length	NULL-Zeichen

Das NULL-Zeichen am Ende des Speicherblocks eines langen Strings wird vom Compiler und den integrierten String-Routinen automatisch verwaltet. Durch das NULL-Zeichen kann ein langer String direkt in einen nullterminierten String umgewandelt werden.

Für String-Konstanten und Literale erzeugt der Compiler einen Speicherblock mit demselben Aufbau wie bei einem dynamisch zugewiesenen String. Der Referenzzähler ist jedoch -1. Wenn Sie einer langen String-Variablen eine String-Konstante zuweisen, wird der String-Zeiger mit der Adresse des Speicherblocks der String-Konstanten belegt. Die integrierten String-Routinen ändern keine Blöcke mit einem Referenzzähler von -1.

Anmerkung: Auf der .NET-Plattform ist `AnsiString` als Array von vorzeichenlosen Bytes implementiert. Die obigen Informationen über String-Konstanten und Literale haben für die .NET-Plattform keine Gültigkeit.

WideString-Typen

In Win32 belegt eine WideString-Variable vier Byte, die einen Zeiger auf einen dynamisch zugewiesenen String enthalten. Wenn eine WideString-Variable leer ist (also einen String der Länge 0 enthält), ist der String-Zeiger **nil**, und der Variablen wird kein dynamischer Speicher zugewiesen. Andernfalls referenziert der String-Zeiger einen dynamisch zugewiesenen Speicherblock,

der neben dem String-Wert eine Längenangabe von 32 Bit enthält. Die folgende Tabelle zeigt den Aufbau eines Speicherblocks für einen WideString in Windows.

Aufbau des Speichers für einen WideString (nur Win32)

Offset	Inhalt
-4	32-Bit-Längenangabe in Byte
0..Länge -1	Zeichen-String
Length	NULL-Zeichen

Die Länge des Strings ergibt sich aus der Anzahl der Byte und ist damit doppelt so groß wie die Anzahl der Wide-Zeichen, die er enthält.

Das NULL-Zeichen am Ende des Speicherblocks eines WideStrings wird vom Compiler und den integrierten String-Routinen automatisch verwaltet. Durch das NULL-Zeichen kann ein WideString direkt in einen nullterminierten String umgewandelt werden.

Auf der .NET-Plattform sind die Typen String und WideString mit dem Typ System.String implementiert. Ein leerer String ist kein **nil**-Zeiger und die String-Daten werden nicht durch ein Null-Zeichen begrenzt.

Mengentypen

Eine Menge ist ein Array von Bits. Jedes Bit zeigt an, ob ein Element in der Menge enthalten ist oder nicht. Da die maximale Anzahl der Elemente einer Menge 256 beträgt, belegt eine Menge nie mehr als 32 Byte. Die Anzahl der Byte, die eine bestimmte Menge belegt, ergibt sich wie folgt: $(Max \text{ div } 8) (Min \text{ div } 8) + 1$, wobei *Max* und *Min* die obere und die untere Grenze des Basistyps der Menge sind. Die Position des Byte eines speziellen Elements *E* ergibt sich wie folgt: $(E \text{ div } 8) (Min \text{ div } 8)$. Die Position des Bit in diesem Byte ergibt sich aus *E* mod 8. Dabei ist *E* der ordinale Wert des Elements. Der Compiler speichert Mengen nach Möglichkeit in CPU-Registern. Eine Menge bleibt jedoch immer im Speicher, wenn sie größer als der generische Integer-Typ ist oder wenn im Quelltext des Programms auf die Adresse der Menge zugegriffen wird.

Anmerkung: Auf der .NET-Plattform sind Mengen, die mehr als 32 Elemente enthalten, als Byte-Arrays implementiert. Der Mengentyp in Delphi für .NET ist nicht CLS-konform, daher kann er von anderen .NET-Sprachen nicht verwendet werden.

Statische Array-Typen

Auf der Win32-Plattform wird ein statisches Array als fortlaufende Folge von Variablen des Komponententyps des Arrays gespeichert. Die Komponenten mit den niedrigsten Indizes werden an der niedrigsten Speicheradresse abgelegt. Bei einem mehrdimensionalen Array liegt die äußerste rechte Dimension an der Basis des belegten Speicherblocks.

Auf der .NET-Plattform ist das statische Array als Typ System.Array implementiert. Das Speicherlayout wird daher vom Typ System.Array bestimmt.

Dynamische Array-Typen

Auf der Win32-Plattform belegt eine Variable für ein dynamisches Array vier Byte, die einen Zeiger auf ein dynamisch zugewiesenes Array enthalten. Wenn eine Variable leer ist (also nicht initialisiert), ist der Zeiger **nil**, und der Variablen wird kein dynamischer Speicher zugewiesen. Andernfalls referenziert die Variable einen dynamisch zugewiesenen Speicherblock, der neben dem Array eine Längenangabe und einen Referenzzähler von je 32 Bit enthält. Die folgende Tabelle zeigt den Aufbau eines Speicherblocks für ein dynamisches Array.

Aufbau des Speichers für ein dynamisches Array (nur Win32)

Offset	Inhalt
-8	32-Bit-Referenzzähler
-4	32-Bit-Längenangabe (Anzahl der Elemente)

0..Länge * (Elementgröße) -1	Array-Elemente
------------------------------	----------------

Auf der .NET-Plattform sind Parameter von dynamischen und offenen Arrays mit dem Typ `System.Array` implementiert. Das Speicherlayout wird daher wie bei statischen Arrays vom Typ `System.Array` bestimmt.

Record-Typen

Auf der .NET-Plattform wird das Felderlayout von Record-Typen zur Laufzeit festgelegt und kann abhängig von der Architektur der Ziel-Hardware variieren. Die folgende Erläuterung der Record-Ausrichtung bezieht sich nur auf die Win32-Plattform (**packed** Records werden aber auf der .NET-Plattform unterstützt).

Wenn ein Record-Typ mit dem voreingestellten Status `{$A+}` deklariert wird und die Deklaration nicht den Modifizierer **packed** enthält, handelt es sich um einen ungepackten Record-Typ. Die Felder des Records werden so ausgerichtet, dass die CPU möglichst effizient darauf zugreifen kann. Die Ausrichtung hängt von den Typen der einzelnen Felder ab und davon, ob Felder zusammen deklariert werden. Jeder Datentyp besitzt eine implizite Ausrichtungsmaske, die vom Compiler automatisch berechnet wird. Sie kann die Werte 1, 2, 4 oder 8 haben und entspricht dem Byte-Raster, in dem ein Wert dieses Typs für den optimalen Zugriff im Speicher angeordnet werden muss. Die folgende Tabelle enthält die Ausrichtungsmasken für alle Datentypen.

Ausrichtungsmasken für Typen (nur Win32)

Typ	Ausrichtung
Ordinale Typen	Größe des Typs (1, 2, 4 oder 8)
Reelle Typen	2 für <code>Real48</code> , 4 für <code>Single</code> , 8 für <code>Double</code> und <code>Extended</code>
Kurze String-Typen	1
Array-Typen	Wie der Typ der Array-Elemente
Record-Typen	Die größte Ausrichtungsmaske der Record-Felder
Mengentypen	Größe des Typs bei 1, 2 oder 4, andernfalls 1
Alle anderen Typen	werden von der <code>\$.A</code> -Direktive festgelegt.

Um die korrekte Ausrichtung der Felder in einem ungepackten Record-Typ zu gewährleisten, fügt der Compiler vor den Feldern mit der Ausrichtungsmaske 2 ein leeres Byte ein. Bei Feldern mit der Ausrichtungsmaske 4 werden nach Bedarf bis zu drei leere Byte eingefügt. Schließlich erweitert der Compiler die gesamte Größe des Records bis zu der Byte-Grenze, die sich aus der größten Ausrichtungsmaske der enthaltenen Felder ergibt.

Wenn die beiden Felder eine gemeinsame Typspezifikation teilen, werden diese selbst dann gepackt, wenn die Deklaration nicht den Modifizierer **packed** enthält und der Record-Typ nicht mit dem `{$A-}`-Status deklariert ist. Betrachten Sie folgende Deklaration:

```
type
  TMyRecord = record
    A, B: Extended;
    C: Extended;
  end;
```

`A` und `B` werden gepackt (an Byte-Grenzen ausgerichtet), da sie dieselben Typspezifikation verwenden. Der Compiler füllt die Struktur mit unbenutzten Bytes aus, um sicherzustellen, dass `C` auf einer Quadword-Grenze erscheint.

Wenn ein Record-Typ mit dem Status `{$A-}` deklariert wird oder die Deklaration den Modifizierer **packed** enthält, werden die Felder des Records nicht ausgerichtet, sondern einfach an aufeinander folgenden Offsets abgelegt. Die Gesamtgröße eines solchen gepackten Records ergibt sich aus der Größe aller Felder. Da sich die Datenausrichtung ändern kann, bieten sich gepackte Record-Typen für alle Record-Strukturen an, die auf Festplatte geschrieben oder im Speicher an ein Modul übergeben werden sollen, das mit einer anderen Version des Compilers compiliert wird.

Dateitypen

Die folgenden Erläuterungen der Dateitypen beziehen sich nur auf die Win32-Plattform. Auf der .NET-Plattform sind Textdateien mit einer Klasse (im Gegensatz zu einem Record) implementiert. Binäre Dateitypen (z.B. `File of MyType`) werden auf der .NET-Plattform nicht unterstützt.

Auf der Win32-Plattform werden Dateitypen als Records repräsentiert. Für jede typisierte und untypisierte Datei wird ein Record mit 332 Byte angelegt, die sich wie folgt verteilen:

```
4
type
TFileRec = packed record
  Handle: Integer;
  Mode: word;
  Flags: word;
  case Byte of
    0: (RecSize: Cardinal);
    1: (BufSize: Cardinal;
        BufPos: Cardinal;
        BufEnd: Cardinal;
        BufPtr: PChar;
        OpenFunc: Pointer;
        InOutFunc: Pointer;
        FlushFunc: Pointer;
        CloseFunc: Pointer;
        UserData: array[1..32] of Byte;
        Name: array[0..259] of Char; );
end;
```

Der Record für Textdateien umfasst 460 Byte, die sich wie folgt verteilen:

```
type
TTextBuf = array[0..127] of Char;
TTextRec = packed record
  Handle: Integer;
  Mode: word;
  Flags: word;
  BufSize: Cardinal;
  BufPos: Cardinal;
  BufEnd: Cardinal;
  BufPtr: PChar;
  OpenFunc: Pointer;
  InOutFunc: Pointer;
  FlushFunc: Pointer;
  CloseFunc: Pointer;
  UserData: array[1..32] of Byte;
  Name: array[0..259] of Char;
  Buffer: TTextBuf;
end;
```

Handle enthält das Handle der Datei (wenn die Datei geöffnet ist).

Das Feld *Mode* kann einen der folgenden Werte annehmen:

```
const
  fmClosed = $D7B0;
  fmInput = $D7B1;
  fmOutput = $D7B2;
  fmInOut = $D7B3;
```

Dabei zeigt *fmClosed* an, dass die Datei geschlossen ist. *fmInput* und *fmOutput* zeigen an, dass es sich bei der Datei um eine Textdatei handelt, die zurückgesetzt (*fmInput*) oder neu geschrieben (*fmOutput*) wurde. *fmInOut* zeigt an, dass die Datei eine typisierte oder eine untypisierte Datei ist, die zurückgesetzt oder neu geschrieben wurde. Jeder andere Wert zeigt an, dass die Dateivariable nicht zugeordnet und damit nicht initialisiert wurde.

Das Feld *UserData* wird für benutzerdefinierte Routinen freigehalten, um Daten zu speichern.

Name enthält den Dateinamen. Dieser besteht aus einer Folge von Zeichen, die mit NULL (#0) abgeschlossen ist.

Bei typisierten und untypisierten Dateien enthält *RecSize* die Record-Länge in Byte. Das Feld *Private* ist reserviert und wird nicht verwendet.

Bei Textdateien ist *BufPtr* ein Zeiger auf einen Puffer mit *BufSize* Byte, *BufPos* ist der Index des nächsten zu lesenden oder zu schreibenden Zeichens des Puffers. *BufEnd* entspricht der Anzahl der gültigen Zeichen im Puffer. *OpenFunc*, *InOutFunc*, *FlushFunc* und *CloseFunc* sind Zeiger auf die E/A-Routinen, welche die Datei verwalten. Weitere Informationen zu diesem Thema finden Sie unter "Gerätetreiberfunktionen". *Flags* legt die Art des Zeilenumbruchs fest:

Bit 0 nicht gesetzt	LF-Zeilenumbruch
Bit 0 gesetzt	CRLF-Zeilenumbruch

Alle anderen Flags-Bits sind für die zukünftige Verwendung reserviert.

Prozedurale Typen

Auf der Win32-Plattform wird ein Prozedurzeiger als 32-Bit-Zeiger auf den Eintrittspunkt einer Prozedur oder Funktion gespeichert. Ein Methodenzeiger wird als 32-Bit-Zeiger auf den Eintrittspunkt einer Methode gespeichert, dem ein 32-Bit-Zeiger auf ein Objekt folgt.

Auf der .NET-Plattform sind prozedurale Typen mit den Klassentypen `System.MulticastDelegate` implementiert.

Klassentypen

Die folgenden Erläuterungen des internen Layouts von Klassentypen beziehen sich nur auf die Win32-Plattform. Auf der .NET-Plattform wird das Klassenlayout zur Laufzeit festgelegt. Laufzeit-Typinformationen werden mittels der APIs `System.Reflection` im .NET-Framework ermittelt.

Auf der Win32-Plattform wird der Wert eines Klassentyps als 32-Bit-Zeiger auf eine Instanz der Klasse gespeichert. Die Instanz einer Klasse wird auch als *Objekt* bezeichnet. Das interne Datenformat eines Objekts gleicht dem eines Records. Die Felder eines Objekts werden in der Reihenfolge ihrer Deklaration als fortlaufende Folge von Variablen gespeichert. Die Felder werden wie bei einem ungepackten Record-Typ immer ausgerichtet. Alle von einer übergeordneten Klasse geerbten Felder werden vor den neuen Feldern gespeichert, die in der abgeleiteten Klasse definiert wurden.

Das erste 4-Byte-Feld eines jeden Objekts ist ein Zeiger auf die *Tabelle der virtuellen Methoden* (VMT) der Klasse. Es gibt nur eine VMT pro Klasse (und nicht eine für jedes Objekt). Zwei verschiedene Klassentypen können eine VMT jedoch nicht gemeinsam benutzen. VMTs werden automatisch vom Compiler erstellt und nie direkt von einem Programm bearbeitet. Ebenso werden die Zeiger auf VMTs automatisch von den Konstruktormethoden in den erstellten Objekten gespeichert und nie direkt von einem Programm bearbeitet.

Die folgende Tabelle zeigt die Struktur einer VMT. Bei positiven Offsets besteht eine VMT aus einer Liste mit 32-Bit-Methodenzeigern. Für jede benutzerdefinierte virtuelle Methode des Klassentyps ist ein Zeiger vorhanden. Die Zeiger sind in der Reihenfolge der Deklaration angeordnet. Jeder Eintrag enthält die Adresse des Eintrittspunktes der entsprechenden virtuellen Methode. Dieses Layout ist zur V-Tabelle von C++ und zu COM kompatibel. Bei negativen Offsets enthält eine VMT die Anzahl der Felder, die in Delphi intern implementiert sind. In einer Anwendung sollten diese Informationen mit den Methoden von `TObject` abgerufen werden, da sich dieses Layout bei künftigen Implementierungen von Delphi ändern kann.

Struktur der Tabelle virtueller Methoden (nur Win32)

Offset	Typ	Beschreibung
-76	Zeiger	Zeiger auf virtuelle Methodentabelle (oder nil)
-72	Zeiger	Zeiger auf Interface-Tabelle (oder nil)
-68	Zeiger	Zeiger auf Informationstabelle zur Automatisierung (oder nil)
-64	Zeiger	Zeiger auf Instanzen-Initialisierungstabelle (oder nil)

4

-60	Zeiger	Zeiger auf Informationstabelle des Typs (oder nil)
-56	Zeiger	Zeiger auf die Tabelle der Felddefinitionen (oder nil)
-52	Zeiger	Zeiger auf die Tabelle der Methodendefinitionen (oder nil)
-48	Zeiger	Zeiger auf die Tabelle der dynamischen Methoden (oder nil)
-44	Zeiger	Zeiger auf einen kurzen String, der den Klassennamen enthält
-40	Cardinal	Instanzgröße in Byte
-36	Zeiger	Zeiger auf einen Zeiger auf die übergeordnete Klasse (oder nil)
-32	Zeiger	Zeiger auf den Eintrittspunkt der Methode <i>Safecall/Exception</i> (oder nil)
-28	Zeiger	Eintrittspunkt der Methode <i>AfterConstruction</i>
-24	Zeiger	Eintrittspunkt der Methode <i>BeforeDestruction</i>
-20	Zeiger	Eintrittspunkt der Methode <i>Dispatch</i>
-16	Zeiger	Eintrittspunkt der Methode <i>DefaultHandler</i>
-12	Zeiger	Eintrittspunkt der Methode <i>NewInstance</i>
-8	Zeiger	Eintrittspunkt der Methode <i>FreeInstance</i>
-4	Zeiger	Eintrittspunkt des Destruktors <i>Destroy</i>
0	Zeiger	Eintrittspunkt der ersten benutzerdefinierten virtuellen Methode
4	Zeiger	Eintrittspunkt der zweiten benutzerdefinierten virtuellen Methode

Klassenreferenztypen

Auf der Win32-Plattform wird Wert einer Klassenreferenz als 32-Bit-Zeiger auf die Tabelle virtueller Methoden (VMT) einer Klasse gespeichert.

Auf der .NET-Plattform sind Klassenreferenztypen mit vom Compiler erstellten, verschachtelten Klassen innerhalb des Klassentyps, den sie unterstützen, implementiert. Diese Implementierungsdetails werden sich in den zukünftigen Compiler-Versionen ändern.

Variante Typen

Die folgenden Erläuterungen des internen Layouts von varianten Typen beziehen sich nur auf die Win32-Plattform. Auf der .NET-Plattform sind Varianten ein Alias von `System.Object`. Varianten sind beim Implementieren von varianten-bezogenen RTL-Funktionen vom Boxing und Unboxing von Daten in einen Objekt-Wrapper sowie von Delphi-Hilfsklassen abhängig.

Auf der Win32-Plattform, wird eine Variante als 16-Byte-Record gespeichert, der einen Typencode und einen Wert (oder eine Referenz auf einen Wert) des durch den Code bezeichneten Typs enthält. Die Units `System` und `Variants` definieren Konstanten und Typen für Varianten.

Der Typ **TVarData** steht für die interne Struktur einer Variante (in Windows entspricht er dem Typ Variant, der von COM und der Win32-API verwendet wird). Mit dem Typ **TVarData** kann bei einer Typumwandlung von Varianten auf die interne Struktur einer Variable zugegriffen werden. Der Record *TVarData* enthält die folgenden Felder::

- *VType* enthält den Typencode der Variante in den niederwertigen zwölf Bits (die Bits, die von der Konstante **varTypeMask** definiert werden). Zusätzlich zeigt das Bit *varArray* an, ob es sich bei der Variante um ein Array handelt. Das Bit *varByRef* gibt an, ob die Variante eine Referenz oder einen Wert enthält.
- Die Felder **Reserved1**, **Reserved2** und **Reserved3** werden nicht verwendet.

Der Inhalt der restlichen acht Bytes eines *TVarData*-Records hängt vom Feld *VType* folgendermaßen ab.

- Wenn keines der *varArray* und *varByRef* gesetzt ist, enthält die Variante einen Wert des gegebenen Typs.
- Wenn *varArray* gesetzt ist, enthält die Variante einen Zeiger auf eine *TVarArray*-Struktur, die das Array definiert. Der Typ

eines jeden Array-Elements ist durch die Bits `varTypeMask` des Feldes `VType` festgelegt.

- Wird das Bit `varByRef` gesetzt, enthält die Variante eine Referenz auf einen Wert des Typs, der durch die Bits `varTypeMask` und `varArray` im Feld `VType` definiert ist.

Der Typencode `varString` ist **private**. Varianten, die einen `varString`-Wert enthalten, sollten nicht an externe Funktionen, also an Funktionen außerhalb von Delphi, übergeben werden. In Win32 sorgt die Automatisierungsunterstützung von Delphi dafür, dass `varString`-Varianten vor der Übergabe an externe Funktionen automatisch in `varOleStr`-Varianten umgewandelt werden.

Siehe auch

Speicherverwaltung auf der Win32-Plattform (siehe Seite 1017)

Speicherverwaltung auf der .NET-Plattform (siehe Seite 1027)

Datentypen (siehe Seite 887)

4.1.1.10.3 Speicherverwaltung auf der .NET-Plattform

Die Laufzeitumgebung von .NET (CLR, Common Language Runtime) ist eine Umgebung mit Speicherbereinigung (Garbage Collection). Das bedeutet, dass sich der Programmierer nicht (oder nur in geringem Umfang) mit der Reservierung und Freigabe von Speicher befassen muss. Die CLR bestimmt, wann eine gefahrlose Freigabe von reserviertem Speicher möglich ist. "Gefahrlos" bedeutet in diesem Zusammenhang, dass keine Referenzen auf den Speicher mehr vorhanden sind.

Dieses Thema enthält folgende Informationen zur Speicherverwaltung:

- Erstellen und Freigeben von Objekten
- Der initialization- und finalization-Abschnitt einer Unit
- Unit-Initialisierung und -Finalisierung in Assemblierungen und Packages

Konstruktoren

In Delphi für .NET muss ein Konstruktor immer einen geerbten Konstruktor aufrufen, bevor er auf geerbte Klassenelemente zugreifen oder diese initialisieren kann (in Delphi für Win32 ist dies nicht der Fall). Fehlt der Aufruf des geerbten Konstruktors, führt dies zu einem Compiler-Fehler. Vor dem Aufruf des geerbten Konstruktors dürfen keinesfalls direkte oder indirekte Aufrufe von geerbten Klassenfeldern erfolgen.

Anmerkung: Ein Konstruktor kann jedoch Felder seiner eigenen Klasse initialisieren, bevor der geerbte Konstruktor aufgerufen wird.

Finalisierung

Im .NET Framework erbt jede Klasse (einschließlich der VCL.NET-Klassen) eine Methode namens `Finalize`. Der Garbage Collector ruft `Finalize` auf, wenn der für das Objekt reservierte Speicher freigegeben werden kann. Der Aufruf erfolgt automatisch, so dass Sie nicht darauf Einfluss nehmen können. Diese asynchrone Finalisierung kann bei Objekten, die Ressourcen wie Datei-Handles oder Datenbankverbindungen öffnen, zu Problemen führen. Wenn die Methode `Finalize` verzögert aufgerufen wird, bleiben diese Verbindungen geöffnet.

Sie können einer Klasse eine Finalisierungsroutine hinzufügen, indem Sie die von `TObject` geerbte, als `strict protected` deklarierte Prozedur `Finalize` überschreiben. Auf der .NET-Plattform wird die Finalisierungsroutine automatisch vom Garbage Collector aufgerufen. Deshalb sind bei ihrer Erstellung gewisse Einschränkungen zu beachten. Unter Umständen wird die Routine nicht in dem Thread ausgeführt, in dem das Objekt erstellt wurde. Eine Finalisierungsroutine kann keinen neuen Speicher reservieren und keine externen Routinen aufrufen. Unterstützt die Klasse Referenzen auf andere Objekte, kann die Finalisierungsroutine diese referenzieren (ihr Speicher ist noch nicht freigegeben). Ihr Status muss aber undefiniert sein, da nicht bekannt ist, ob bereits eine Finalisierung für sie erfolgt ist.

Wenn für eine Klasse eine Finalisierungsroutine definiert ist, muss die CLR neu instantiierte Objekte der Klasse der Finalisierungsliste hinzufügen. Objekte, für die Finalisierungsroutinen existieren, bleiben in der Regel länger im Speicher. Der

Garbage Collector gibt sie nicht sofort frei, wenn er zum ersten Mal feststellt, dass keine aktiven Referenzen mehr auf sie vorhanden sind. Referenziert ein Objekt andere Objekte, verbleiben diese im Speicher, bis das Originalobjekt freigegeben wird (auch dann, wenn für die betreffenden Objekte keine eigenen Finalisierungsroutinen vorhanden sind). Da Finalisierungsroutinen aus den genannten Gründen wesentlich zum Speicherverbrauch (und damit zur Verringerung der Ausführungsgeschwindigkeit) beitragen, sollten sie sehr zielgerichtet eingesetzt werden.

Es ist gängige Programmierpraxis, Finalisierungsroutinen auf kleine Objekte zu beschränken, die nicht verwaltete Ressourcen repräsentieren. Klassen, die diese Ressourcen nutzen, können dann per Referenz auf das kleine Objekt mit zugehöriger Finalisierungsroutine zugreifen. Auf diese Weise wird verhindert, dass große Klassen (und solche, die viele andere Klassen referenzieren), aufgrund der Finalisierungsroutine sehr viel Speicher belegen.

Die Ausführung von Finalisierungsroutinen sollte außerdem verhindert werden, wenn eine bestimmte Ressource bereits in einem Destruktor freigegeben wurde. Hierfür steht die Methode `SuppressFinalize` zur Verfügung. Sie sollte nach der Freigabe von Ressourcen aufgerufen werden und veranlasst, dass die CLR das jeweilige Objekt aus der Finalisierungsliste löscht. `SuppressFinalize` darf keinesfalls mit einer `nil`-Referenz aufgerufen werden, da dies zu einer Laufzeit-Exception führen würde.

Das Dispose-Muster

Ein andere Möglichkeit zur Freigabe von Ressourcen besteht in der Implementierung des *Dispose-Musters*. Klassen, die das Dispose-Muster verwenden, müssen das .NET-Interface `IDisposable` implementieren. `IDisposable` enthält nur die Methode `Dispose`. Im Gegensatz zu `Finalize` ist die Methode `Dispose` als `public` deklariert. Sie setzt keinen Aufruf durch den Garbage Collector voraus, sondern kann direkt aufgerufen werden. Sie haben damit die Möglichkeit, die Freigabe von Ressourcen zu steuern. `Dispose` gibt aber nicht den Speicher für das Objekt selbst frei. Dies ist weiterhin Aufgabe des Garbage Collector. Einige Klassen im .NET Framework implementieren neben `Dispose` eine weitere Methode, z. B. `Close`. `Close` veranlasst zwar nur einen Aufruf von `Dispose`, ist aber für bestimmte Klassen (etwa für die Dateiverarbeitung) geeigneter.

Delphi für .NET unterstützt zwar die Methode `Finalize` zur Freigabe von Systemressourcen, grundsätzlich sollte aber der Implementierung des Dispose-Musters der Vorzug gegeben werden. Der Delphi .NET-Compiler erkennt ein spezielles Destruktor-Muster in einer Klasse und sorgt für eine entsprechende Implementierung des `IDisposable`-Interface. Sie können neuen Quelltext für die .NET-Plattform deshalb wie gewohnt schreiben. Gleichzeitig ist es möglich, vorhandenen Delphi-Code für Win32 in der Speicherbereinigungsumgebung der CLR auszuführen.

Der Compiler erkennt das folgende Muster eines Delphi-Destruktors:

```
TMyClass = class(TObject)
  destructor Destroy; override;
end;
```

Das Muster muss exakt in dieser Form auf den Destruktor angewendet werden:

- Der Name des Destruktors muss `Destroy` lauten.
- Das Schlüsselwort `override` muss angegeben werden.
- Der Destruktor kann keine Parameter übernehmen.

Der Compiler setzt das Dispose-Muster folgendermaßen um: Die Methode `Free` wird so definiert, dass ein Aufruf der Methode `Dispose` erfolgt, wenn die Klasse das Interface `IDisposable` implementiert (und dies ist der Fall). `Dispose` veranlasst dann den Aufruf des von Ihnen bereitgestellten Konstruktors.

Sie können das Interface `IDisposable` zwar auch weiterhin direkt implementieren, die automatische Implementierung des Free-Dispose-Destroy-Mechanismus des Compilers kann dann aber nicht genutzt werden. Die beiden Verfahren zur Implementierung von `IDisposable` schließen sich gegenseitig aus. Sie haben also die Wahl zwischen einer direkten Implementierung von `IDisposable` und der Verwendung des `destructor Destroy; override`-Musters, bei dem der Compiler alle restlichen Aufgaben übernimmt. In beiden Fällen ruft `Free` die Methode `Dispose` auf. Wenn Sie jedoch eine eigene Implementierung von `Dispose` bereitstellen, müssen Sie selbst für den Aufruf des Destruktors sorgen. Wenn Sie `IDisposable` direkt implementieren, dürfen Sie den Destruktor nicht `Destroy` nennen.

Anmerkung: Sie können bei der Deklaration von Destruktoren eigene Namen vergeben. Der Compiler unterstützt die `IDisposable`-Implementierung aber nur dann, wenn der Destruktorkonstruktor dem obigen Muster entspricht.

Die Methode `Dispose` wird nicht automatisch aufgerufen. Der Aufruf von `Dispose` erfolgt erst, nachdem die Methode `Free` aufgerufen wurde. Sind keine Referenzen mehr auf ein Objekt vorhanden, wird es vom Garbage Collector freigegeben. Wenn vorher jedoch kein expliziter Aufruf der Methode `Free` für das Objekt erfolgt ist, unterbleibt die Ausführung des Destruktors.

Anmerkung: Wenn der Garbage Collector den von einem Objekt belegten Speicher freigibt, werden auch alle Felder der Objektinstanz freigegeben. Der gebräuchlichste Grund für die Implementierung von Destruktoren in Delphi für Win32 - die Freigabe des reservierten Speichers - ist also nicht mehr gegeben. Nicht verwaltete Ressourcen, wie Fenster- oder Datei-Handles, müssen in der Regel aber weiterhin freigegeben werden.

Um den mehrfachen Aufruf von Destruktoren zu verhindern, unterstützt der Delphi .NET-Compiler in allen Klassendeklarationen ein neues Feld namens `DisposeCount`. Existiert in einer Klasse bereits ein Feld dieses Namens, führt der Namenskonflikt zu einem Syntaxfehler im Destruktorkonstruktor.

Der initialization- und finalization-Abschnitt einer Unit

Auf der .NET-Plattform werden alle Units, auf die Bezug genommen wird, vor der eigenen Unit initialisiert. Reihenfolge und Zeitpunkt der Initialisierung können jedoch nicht exakt vorherbestimmt werden. Achten Sie speziell auf Initialisierungscode, der von den Nebeneffekten der Initialisierung einer anderen Unit abhängig ist, etwa der Erstellung einer Datei. Derartige Abhängigkeiten lassen sich nicht so umsetzen, dass sie auf der .NET-Plattform zuverlässig funktionieren.

Für die Finalisierung einer Unit gelten dieselben Beschränkungen wie für die `Finalize`-Methode von Objekten. Sie erfolgt asynchron, und es kann nicht im Voraus bestimmt werden, wann sie durchgeführt wird.

Zur Finalisierung einer Unit gehört das Freigeben globaler Objekte, das Aufheben der Registrierung von Objekten, die von anderen Units verwendet werden, sowie das Freigeben von Ressourcen. Da .NET eine Umgebung mit Speicherbereinigung ist, gibt der Garbage Collector globale Objekte auch dann frei, wenn der **finalization**-Abschnitt der Unit nicht aufgerufen wird. Die Units in einer Anwendungsdomäne werden zusammen ge- und entladen. Sie müssen sich deshalb nicht um die Aufhebung der Registrierung von Objekten kümmern. Alle Units, bei denen die Möglichkeit einer gegenseitigen Referenzierung besteht (auch in unterschiedlichen Assemblierungen), werden zum selben Zeitpunkt freigegeben. Da Objektreferenzen nicht über Anwendungsdomänen hinweg möglich sind, besteht nicht die Gefahr einer verbleibenden Referenz auf einen Objekttyp oder auf Code, der aus dem Speicher entfernt wurde.

Die Freigabe von Ressourcen (wie Datei- oder Fenster-Handles) ist die wichtigste Aufgabe bei der Finalisierung einer Unit. Da nicht sicher ist, dass der **finalization**-Abschnitt einer Unit aufgerufen wird, kann eine Überarbeitung des vorhandenen Quelltexts und eine Implementierung von Finalisierungsroutinen erforderlich sein.

Folgende Punkte sind bei der Initialisierung und Finalisierung von Units auf der .NET-Plattform besonders zu beachten:

1. Der Aufruf der Methode `Finalize` erfolgt asynchron (sowohl für Objekte als auch für Units).
2. Mittels Finalisierung und Destruktoren werden nicht verwaltete Ressourcen wie Datei-Handles freigegeben. Die Freigabe von Objektvariablen ist nicht erforderlich. Dies erfolgt automatisch durch den Garbage Collector.
3. Klassen sollten die `IDisposable`-Implementierung des Compilers verwenden und einen Destruktorkonstruktor mit dem Namen `Destroy` bereitstellen.
4. Eine Klasse, die `IDisposable` direkt implementiert, darf keinen Destruktorkonstruktor mit dem Namen `Destroy` enthalten.
5. Die Referenzzählung ist veraltet. Wenn möglich, sollte immer das Muster `destructor Destroy; override;` verwendet werden.
6. Die Unit-Initialisierung darf nicht von Nebeneffekten abhängig sein, die durch die Initialisierung anderer Units entstehen.

Unit-Initialisierung für Assemblierungen und dynamisch gelinkte Packages

In Win32 nutzt der Delphi-Compiler die Funktion `DllMain` als Hook für die Ausführung des Unit-Initialisierungscodes. In .NET steht kein entsprechender Ausführungspfad zur Verfügung. Das .NET Framework bietet dafür andere Tools für die Implementierung der Unit-Initialisierung. Die Implementierungsunterschiede zwischen Win32 und .NET können sich allerdings

auf die Reihenfolge der Unit-Initialisierung einer Anwendung auswirken.

Der Delphi .NET-Compiler stellt Hooks für die Unit-Initialisierung mittels CLS-konformen Klassenkonstruktoren bereit. Die CLR fordert, dass für jeden Objekttyp ein eigener Klassenkonstruktor vorhanden ist. Diese Konstruktoren, auch Typinitialisierer genannt, werden *mindestens* einmal ausgeführt. Dies ist erforderlich, damit der Typ geladen wird. Die Assemblierung, die einen Typ enthält, wird erst geladen, wenn dieser Typ zur Laufzeit verwendet wird. Unterbleibt das Laden der Assemblierung, wird der **initialization**-Abschnitt für die Unit nie ausgeführt.

Zirkuläre Unit-Referenzen wirken sich ebenfalls auf die Unit-Initialisierung aus. Wenn Unit A Unit B referenziert und diese dann wieder Unit A in ihrem **implementation**-Abschnitt enthält, ist die Reihenfolge für die Unit-Initialisierung nicht definiert. Ein Blick auf die einzelnen Schritte dieses Prozesses verdeutlicht, warum.

1. Der **initialization**-Abschnitt von Unit A referenziert einen Typ aus Unit B. Handelt es sich dabei um die erste Referenz auf den Typ, lädt die CLR die zugehörige Assemblierung und löst die Initialisierung von Unit B aus.
2. Folglich wird Unit B geladen und initialisiert, bevor der **initialization**-Abschnitt von Unit A vollständig ausgeführt ist. Die Unit-Initialisierung funktioniert hier also anders als unter Win32.
3. Angenommen, es wird ein Typ aus Unit A verwendet, während die Initialisierung von Unit B gerade läuft. Da die Initialisierung von Unit A noch nicht abgeschlossen ist, kann diese Referenz zu einer Zugriffsverletzung führen.

Bei der Unit-Initialisierung sollten nur Typen verwendet werden, die in dieser Unit definiert sind. Die Referenzierung von Typen außerhalb der Unit kann deren Initialisierung behindern und (wie oben beschrieben) zu einer Zugriffsverletzung führen.

Die Unit-Initialisierung für DLLs erfolgt automatisch. Sie wird ausgelöst, wenn ein Typ innerhalb der DLL referenziert wird. Anwendungen, die mit anderen .NET-Sprachen programmiert wurden, können Assemblierungen von Delphi für .NET nutzen, ohne dass Probleme bei der Unit-Initialisierung entstehen.

Siehe auch

Speicherverwaltung auf der Win32-Plattform (siehe Seite 1017)

Interne Datenformate (siehe Seite 1018)

4.1.1.11 Ablaufsteuerung

In diesem Abschnitt wird die Übergabe von Parametern an Prozeduren und Funktionen beschrieben.

4.1.1.11.1 Ablaufsteuerung

Bevor Sie mit der Programmierung von Anwendungen beginnen, müssen Sie wissen, wie Parameter übergeben und Funktionsergebnisse verarbeitet werden. Die Übergabe ist von verschiedenen Faktoren abhängig. Dazu gehören die Aufrufkonventionen, die Parametersemantik sowie der Typ und die Größe des zu übergebenden Wertes.

Dieses Thema enthält Informationen zu folgenden Bereichen:

- Parameterübergabe
- Funktionsergebnisse
- Methodenaufrufe
- Exit-Prozeduren

Parameterübergabe

Die Übergabe von Parametern an Prozeduren und Funktionen erfolgt entweder über CPU-Register oder über den Stack. Welche Übergabemethode verwendet wird, hängt von der Aufrufkonvention der Routine ab. Informationen über Aufrufkonventionen finden Sie im Thema "Aufrufkonventionen".

Übergabe per Wert oder per Referenz

Variablenparameter (**var**) werden immer per Referenz übergeben, also als 32-Bit-Zeiger auf die tatsächliche Speicherposition.

Wert- und Konstantenparameter (**const**) werden abhängig vom Typ und der Größe des Parameters als Wert oder als Referenz übergeben:

- Ein Parameter ordinalen Typs wird als 8-Bit-, 16-Bit-, 32-Bit- oder 64-Bit-Wert übergeben. Dabei wird dasselbe Format verwendet wie bei einer Variable des entsprechenden Typs.
- Ein Parameter reellen Typs wird immer im Stack übergeben. Ein *Single*-Parameter benötigt vier Byte, ein *Double*-, *Comp*- oder *Currency*-Parameter belegt acht Byte. Auch ein *Real48*-Parameter belegt acht Byte. Dabei wird der *Real48*-Wert in den niedrigen sechs Byte gespeichert. Ein *Extended*-Parameter belegt zwölf Byte, wobei der *Extended*-Wert in den niedrigen zehn Byte gespeichert wird.
- Ein kurzer String-Parameter wird als 32-Bit-Zeiger auf einen kurzen String übergeben.
- Lange String-Parameter und dynamische Array-Parameter werden als 32-Bit-Zeiger auf den dynamischen Speicherblock übergeben, der für den langen String reserviert wurde. Für einen leeren langen String wird der Wert **nil** übergeben.
- Ein Zeiger, eine Klasse, eine Klassenreferenz oder ein Prozedurzeiger wird als 32-Bit-Zeiger übergeben.
- Ein Methodenzeiger wird immer als zwei 32-Bit-Zeiger im Stack übergeben. Der Instanzzeiger wird vor dem Methodenzeiger auf dem Stack abgelegt, so dass der Methodenzeiger die niedrigere Adresse erhält.
- Mit den Konventionen **register** und **pascal** wird ein variabler Parameter als 32-Bit-Zeiger auf einen Variant-Wert übergeben.
- Mengen, Records und statische Arrays aus einem, zwei oder vier Byte werden als 8-Bit-, 16-Bit- und 32-Bit-Werte übergeben. Größere Mengentypen, Records und statische Arrays werden als 32-Bit-Zeiger auf den Wert übergeben. Eine Ausnahme von dieser Regel ist, dass bei den Konventionen **cdecl**, **stdcall** und **safecall** die Records immer direkt im Stack übergeben werden. Die Größe eines auf diese Weise übergebenen Records wird immer bis zur nächsten Double-Word-Grenze erweitert.
- Ein offener Array-Parameter wird in Form zweier 32-Bit-Werte übergeben. Der erste Wert ist ein Zeiger auf die Array-Daten. Der zweite Wert enthält die Anzahl der Array-Elemente minus eins.

Bei der Übergabe zweier Parameter im Stack belegt jeder Parameter immer ein Vielfaches von vier Byte (also eine ganzzahlige Anzahl von Double Words). Ein 8-Bit- oder 16-Bit-Parameter wird auch dann als Double Word übergeben, wenn er nur ein Byte oder ein Word belegt. Der Inhalt der nicht verwendeten Byte des Double Word ist nicht definiert.

Die Konventionen **pascal**, **cdecl**, **stdcall** und **safecall**

Bei Verwendung der Konventionen **pascal**, **cdecl**, **stdcall** und **safecall** werden alle Parameter im Stack übergeben. Bei der Konvention **pascal** werden die Parameter in der Reihenfolge ihrer Deklaration (von links nach rechts) übergeben, so dass der erste Parameter im Stack an der obersten Adresse und der letzte Parameter an der untersten Adresse gespeichert wird. Bei den Konventionen **cdecl**, **stdcall** und **safecall** werden die Parameter in der entgegengesetzten Reihenfolge ihrer Deklaration (von rechts nach links) übergeben, so dass der erste Parameter im Stack an der untersten Adresse und der letzte an der obersten Adresse gespeichert wird.

Die Konvention **register**

Bei der Konvention **register** werden maximal drei Parameter in den CPU-Registern übergeben, der Rest im Stack. Die Parameter werden in der Reihenfolge ihrer Deklaration übergeben (wie bei der Konvention **pascal**). Die ersten drei geeigneten Parameter stehen in den Registern EAX, EDX und ECX (in dieser Reihenfolge). Nur reelle, variante und strukturierte Typen sowie Methodenzeiger- und Int64-Typen sind als Registerparameter ungeeignet. Sind mehr als drei mögliche Registerparameter vorhanden, werden die ersten drei in EAX, EDX und ECX übergeben. Die restlichen Parameter werden in der Reihenfolge ihrer Deklaration im Stack abgelegt. Betrachten Sie beispielsweise die folgende Deklaration:

```
procedure Test(A: Integer; var B: Char; C: Double; const D: string; E: Pointer);
```

Hier übergibt die Prozedur *Test* den Parameter A in EAX als 32-Bit-Integer, B in EDX als Zeichenzeiger und D in ECX als Zeiger auf einen Speicherblock für einen langen String. C wird im Stack in Form zweier Double Words und E als 32-Bit-Zeiger (in dieser Reihenfolge) abgelegt.

Konventionen zur Speicherung in Registern

Prozeduren und Funktionen dürfen die Register EBX, ESI, EDI und EBP nicht verändern. Die Register EAX, EDX und ECX stehen jedoch zur Verfügung. Wenn ein Konstruktor oder Destruktor in Assembler implementiert wird, muss das DL-Register unverändert bleiben. Prozeduren und Funktionen werden unter der Voraussetzung aufgerufen, dass das Richtungsflag der CPU nicht gesetzt ist (entsprechend einer CLD-Anweisung). Auch nach Beendigung der Routine darf das Richtungsflag nicht gesetzt sein.

Anmerkung: Delphi-Prozeduren und -Funktionen werden in der Regel unter der Voraussetzung aufgerufen, dass der CPU-Stack leer ist. Der Compiler versucht bei der Generierung von Code alle acht FPU-Stackeinträge zu verwenden.

Achten Sie bei der Arbeit mit MMX- und XMM-Anweisungen darauf, dass die Werte des xmm- und mm-Registers erhalten bleiben. Delphi-Funktionen werden unter der Voraussetzung aufgerufen, dass die x87 FPU-Datenregister für die x87-Gleitkommaanweisungen zur Verfügung stehen. Der Compiler setzt also voraus, dass die EMMS/FEMMS-Anweisungen nach den MMX-Operationen aufgerufen wurden. Delphi-Funktionen stellen keine Anforderungen an den Status und Inhalt von xmm-Registern. Sie gewährleisten auch nicht, dass der Inhalt der xmm-Register unverändert bleibt.

Funktionsergebnisse

Für die Rückgabe von Funktionsergebnissen gelten folgende Konventionen.

- Funktionsergebnisse ordinalen Typs werden, wenn möglich, in ein CPU-Register zurückgegeben. Bytes werden in AL, Words in AX und Double Words in EAX zurückgegeben.
- Die Funktionsergebnisse der Real-Typen werden im Top-of-Stack-Register des Coprozessors für Gleitkommazahlen (ST(0)) zurückgegeben. Bei Funktionsergebnissen vom Typ *Currency* wird der Wert von ST(0) um den Faktor 10000 skaliert. Beispielsweise wird der *Currency*-Wert 1,234 in ST(0) als 12340 zurückgegeben.
- Strings, dynamische Arrays, Methodenzeiger oder Varianten werden so zurückgegeben, als ob das Funktionsergebnis als zusätzlicher **var**-Parameter nach den übrigen Parametern deklariert worden wäre. Die aufrufende Routine übergibt also einen zusätzlichen 32-Bit-Zeiger auf eine Variable, über die das Funktionsergebnis zurückgeliefert wird.
- Int64 wird in EDX:EAX zurückgegeben.
- Zeiger, Klassen, Klassenreferenzen und Prozedurzeiger werden in EAX zurückgegeben.
- Statische Arrays, Records und Mengen werden in AL zurückgegeben, wenn der Wert ein Byte belegt, in AX, falls der Wert zwei Byte belegt, und in EAX, falls vier Byte benötigt werden. Andernfalls wird der Funktion nach den deklarierten Parametern ein zusätzlicher **var**-Parameter übergeben, über den die Funktion das Ergebnis zurückliefert.

Methodenaufrufe

Für Methoden werden dieselben Aufrufkonventionen wie für normale Prozeduren und Funktionen verwendet. Jede Methode verfügt jedoch über den zusätzlichen Parameter *Self*. Dabei handelt es sich um eine Referenz auf die Instanz oder Klasse, in der die Methode aufgerufen wird. Der Parameter *Self* wird als 32-Bit-Zeiger übergeben.

- Bei der Konvention **register** verhält sich der Parameter *Self*, als ob er vor allen anderen Parametern deklariert worden wäre. Er wird somit immer im Register EAX übergeben.
- Bei der Konvention **pascal** verhält sich der Parameter *Self*, als ob er nach allen anderen Parametern (einschließlich des zusätzlichen **var**-Parameters für das Funktionsergebnis) deklariert worden wäre. Er wird somit als letzter Parameter übergeben und hat von allen Parametern die niedrigste Adresse.
- Bei den Konventionen **cdecl**, **stdcall** und **safecall** verhält sich der Parameter *Self*, als ob er vor allen anderen Parametern, aber nach dem zusätzlichen **var**-Parameter für das Funktionsergebnis deklariert worden wäre. Er wird daher als letzter Parameter, aber vor dem zusätzlichen **var**-Parameter (falls vorhanden) übergeben.

Konstruktoren und Destruktoren verwenden dieselben Aufrufkonventionen wie die anderen Methoden. Es wird jedoch ein zusätzlicher Boolescher Flag-Parameter übergeben, der den Kontext des Konstruktor- oder Destruktoraufrufs anzeigt.

Der Wert *False* im Flag-Parameter eines Konstruktoraufrufs zeigt an, dass der Konstruktor über ein Instanzobjekt oder mit dem Schlüsselwort **inherited** aufgerufen wurde. In diesem Fall verhält sich der Konstruktor wie eine normale Methode. Der Wert *True*

im Flag-Parameter eines Konstruktorauftrufs zeigt an, dass der Konstruktor über eine Klassenreferenz aufgerufen wurde. In diesem Fall erzeugt der Konstruktor eine Instanz der mit *Self* referenzierten Klasse und gibt in EAX eine Referenz auf das neu erzeugte Objekt zurück.

Der Wert *False* im Flag-Parameter eines Destruktorauftrufs zeigt an, dass der Destruktork mit dem Schlüsselwort **inherited** aufgerufen wurde. In diesem Fall verhält sich der Destruktork wie eine normale Methode. Der Wert *True* im Flag-Parameter eines Destruktorauftrufs zeigt an, dass der Destruktork über ein Instanzobjekt aufgerufen wurde. In diesem Fall gibt der Destruktork die mit *Self* bezeichnete Instanz frei, bevor er beendet wird.

Der Flag-Parameter verhält sich so, als ob er vor allen anderen Parametern deklariert worden wäre. Bei der Konvention **register** wird er im Register DL übergeben. Bei **pascal** erfolgt die Übergabe vor allen anderen Parametern. Bei den Konventionen **cdecl**, **stdcall** und **safecall** wird er direkt vor dem Parameter *Self* übergeben.

Der Wert von DL muss deshalb vor der Beendigung des Programms wiederhergestellt werden, damit *BeforeDestruction* oder *AfterConstruction* ordnungsgemäß aufgerufen werden kann.

Exit-Prozeduren

Mit Exit-Prozeduren können Sie sicherstellen, dass vor Beendigung eines Programms bestimmte Aktionen (z. B. das Aktualisieren und Schließen von Dateien) eingeleitet werden. Mithilfe der Zeigervariablen *ExitProc* kann eine Exit-Prozedur *installiert* werden, die bei jeder Beendigung des Programms aufgerufen wird. Dabei ist es gleichgültig, ob das Programm normal, über einen Aufruf von *Halt* oder aufgrund eines Laufzeitfehlers beendet wird. Einer Exit-Prozedur werden keine Parameter übergeben.

Anmerkung: Es empfiehlt sich, alle Abläufe bei der Programmbeendigung nicht über Exit-Prozeduren, sondern über **finalization**-Abschnitte zu steuern. Exit-Prozeduren können nur für ausführbare Dateien verwendet werden. Für DLLs (Win32) können Sie eine ähnliche Variable verwenden: Sie heißt *DllProc* und wird sowohl beim Laden als auch beim Entladen der Bibliothek aufgerufen. Bei der Verwendung von Packages muss das gewünschte Verhalten in einem **finalization**-Abschnitt implementiert werden. Alle Exit-Prozeduren werden aufgerufen, bevor die **finalization**-Abschnitte ausgeführt werden.

Exit-Prozeduren können von Units und von Programmen installiert werden. Eine Unit kann eine Exit-Prozedur im **initialization**-Abschnitt installieren. Die Prozedur ist dann für die erforderlichen Aufräumarbeiten verantwortlich (z. B. das Schließen von Dateien).

Bei korrekter Implementierung ist jede Exit-Prozedur ein Glied in einer Kette von Exit-Prozeduren. Alle Prozeduren in der Kette werden in der umgekehrten Reihenfolge ihrer Installation ausgeführt. Dadurch ist sichergestellt, dass der Beendigungscode einer bestimmten Unit nicht vor dem Beendigungscode der Units ausgeführt wird, die von ihr abhängen. Um die Beendigungskette nicht zu unterbrechen, müssen Sie den aktuellen Inhalt von *ExitProc* speichern, bevor Sie ihr die Adresse Ihrer eigenen Beendigungsprozedur zuweisen. Außerdem muss der gespeicherte Wert von *ExitProc* in der ersten Anweisung Ihrer Beendigungsprozedur wiederhergestellt werden.

Das folgende Beispiel skizziert die Implementierung einer solchen Prozedur:

```
var
  ExitSave: Pointer;

procedure MyExit;

begin
  ExitProc := ExitSave; // Zuerst immer den alten Vektor wiederherstellen .
  .
  .
end;

begin
  ExitSave := ExitProc;
  ExitProc := @MyExit;
  .

```

end.

Zuerst wird der Inhalt von `ExitProc` in `ExitSave` gespeichert. Anschließend wird die Prozedur `MyExit` installiert. Nachdem die Prozedur als Teil des Beendigungsvorgangs aufgerufen wurde, wird mit `MyExit` zuerst die bisherige Exit-Prozedur wiederhergestellt.

Die Beendigungsroutine der Laufzeitbibliothek ruft Exit-Prozeduren auf, bis `ExitProc` den Wert `nil` annimmt. Um Endlosschleifen zu vermeiden, wird `ExitProc` vor jedem Aufruf auf `nil` gesetzt. Die nächste Exit-Prozedur wird somit nur aufgerufen, wenn `ExitProc` in der aktuellen Exit-Prozedur eine Adresse zugewiesen wird. Tritt in einer Exit-Prozedur ein Fehler auf, wird sie nicht erneut aufgerufen.

Eine Exit-Prozedur kann die Ursache einer Beendigung feststellen, indem sie die Integer-Variable `ExitCode` und die Zeigervariable `ErrorAddr` auswertet. Bei einer normalen Beendigung hat `ExitCode` den Wert Null und `ErrorAddr` den Wert `nil`. Wird ein Programm durch einen Aufruf von `Halt` beendet, enthält `ExitCode` den der Funktion `Halt` übergebenen Wert und `ErrorAddr` den Wert `nil`. Wird das Programm aufgrund eines Laufzeitfehlers beendet, enthält `ExitCode` den Fehlercode und `ErrorAddr` die Adresse der ungültigen Anweisung.

Die letzte Exit-Prozedur (die von der Laufzeitbibliothek installiert wird) schließt die Ein- und Ausgabedateien. Hat `ErrorAddr` nicht den Wert `nil`, wird eine Meldung über den Laufzeitfehler ausgegeben. Wenn Sie selbst Meldungen zu Laufzeitfehlern ausgeben wollen, installieren Sie eine Exit-Prozedur, die `ErrorAddr` auswertet und eine Meldung ausgibt, wenn die Variable nicht den Wert `nil` hat. Zusätzlich müssen Sie vor dem Ende der Prozedur den Wert von `ErrorAddr` auf `nil` setzen, so dass der Fehler nicht in anderen Exit-Prozeduren erneut ausgegeben wird.

Nachdem die Laufzeitbibliothek alle Exit-Prozeduren aufgerufen hat, wird die Steuerung an das Betriebssystem zurückgegeben und der in `ExitCode` gespeicherte Wert als Rückgabewert übergeben.

Siehe auch

Prozeduren und Funktionen (siehe Seite 931)

4.1.1.12 Verwendung des integrierten Assemblers (nur Win32)

Dieser Abschnitt befasst sich mit der Verwendung des integrierten Assemblers auf der Win32-Plattform.

4.1.1.12.1 Verwendung des integrierten Assemblers (nur Win32)

Mit dem integrierten Assembler können Sie Assembler-Code direkt in Ihre Delphi-Programme einfügen. Der integrierte Assembler ist nur im Win32-Compiler von Delphi verfügbar. Er besitzt die folgenden Funktionsmerkmale:

- Inline-Assemblierung
- Unterstützung aller Anweisungen in Intel Pentium 4, Intel MMX Extensions, Streaming SIMD Extensions (SSE) sowie AMD Athlon (einschließlich 3D Now!)
- Keine Makrounterstützung, ermöglicht jedoch reine Assembler-Prozeduren
- Verwendung von Delphi-Bezeichnern, wie etwa Konstanten, Typen und Variablen in Assembler-Anweisungen

Alternativ zur Verwendung des integrierten Assemblers können Sie Objektdateien hinzulinken, die als `external` deklarierte Prozeduren und Funktionen enthalten. Weitere Informationen finden Sie im Thema zu `external`-Deklarationen. Wenn Sie in einer Anwendung externen Assembler-Quellcode verwenden möchten, sollten Sie ihn in Delphi neu erstellen oder zumindest mit dem integrierten Assembler neu implementieren.

Die Anweisung `asm`

Auf den integrierten Assembler greifen Sie über `asm`-Anweisungen zu, die folgende Syntax haben:

```
asm Anweisungsliste end
```

Dabei steht **Anweisungsliste** für eine Folge von Assembler-Anweisungen, die durch Strichpunkte, Zeilenendezeichen oder Delphi-Kommentare voneinander getrennt werden.

Kommentare in einer **asm**-Anweisung müssen dem Delphi-Stil entsprechen. Ein Strichpunkt besagt hier nicht, dass es sich beim Rest der Zeile um einen Kommentar handelt.

Das reservierte Wort **inline** und die Direktive **assembler** werden aus Gründen der Abwärtskompatibilität beibehalten. Sie haben keine Auswirkungen auf den Compiler.

Mit Registern arbeiten

Im Allgemeinen sind die Regeln für die Verwendung von Registern in einer **asm**-Anweisung identisch mit denjenigen für eine **external**-Prozedur oder -Funktion. In einer **asm**-Anweisung muss der Inhalt der Register EDI, ESI, ESP, EBP und EBX erhalten bleiben, während die Register EAX, ECX und EDX beliebig geändert werden können. Beim Eintritt in eine **asm**-Anweisung zeigt EBP auf den aktuellen Stackframe, ESP auf den Beginn des Stacks. Zu Beginn der Ausführung einer **asm**-Anweisung ist der Registerinhalt unbekannt. Eine Ausnahme bilden die Register ESP und EBP.

Siehe auch

Assembler-Syntax (siehe Seite 1035)

Assembler-Ausdrücke (siehe Seite 1040)

Assembler-Prozeduren und -Funktionen (siehe Seite 1048)

4.1.1.12.2 Assembler-Syntax (nur Win32)

Der integrierte Assembler ist nur im Win32-Compiler von Delphi verfügbar. Im Folgenden werden die Elemente der Assembler-Syntax erläutert, die für eine ordnungsgemäße Verwendung unabdingbar sind.

- Syntax für Assembler-Anweisungen
- Labels
- Anweisungs-Opcodes
- Assembler-Direktiven
- Operanden

Syntax für Assembler-Anweisungen

Die Syntax für eine Assembler-Anweisung lautet:

Label: Präfix *Opcode* *Operand1*, *Operand2*

Label steht für einen Label-Bezeichner, *Präfix* für einen Assembler-Präfix-Opcode (Operationscode), *Opcode* für eine Assembler-Anweisung oder -Direktive und *Operand* für einen Assembler-Ausdruck. *Label* und *Präfix* sind optional. Es gibt Opcodes mit nur einem Operanden, während andere überhaupt keine Operanden haben.

Kommentare sind nur zwischen, nicht aber innerhalb von Assembler-Anweisungen erlaubt. Beispiel:

```
MOV AX,1 {Anfangswert} { OK }
MOV CX,100 {Count} { OK }
MOV {Anfangswert} AX,1; { Fehler! }
MOV CX, {Count} 100 { Fehler! }
```

Labels

Label werden im integrierten Assembler auf die gleiche Weise wie in Delphi definiert. Geben Sie vor einer Anweisung ein Label und einen Doppelpunkt ein. Für ein Label gibt es keine Längenbeschränkung. Wie in Delphi müssen auch in Assembler alle Label im **label**-Deklarationsabschnitt des Blocks definiert werden, der die **asm**-Anweisung enthält. Von dieser Regel gibt es eine Ausnahme: lokale Label.

Lokale Label beginnen immer mit dem Zeichen @. Danach können ein oder mehrere Buchstaben, Ziffern, Unterstriche oder @-Zeichen angegeben werden. Ein lokales Label ist auf **asm**-Anweisungen beschränkt. Der Gültigkeitsbereich eines lokalen Label erstreckt sich vom Schlüsselwort **asm** bis zum Ende der **asm**-Anweisung, in der sich das Label befindet. Ein lokales Label braucht nicht deklariert zu werden.

Anweisungs-Opcodes

Der integrierte Assembler unterstützt alle von Intel dokumentierten Opcodes. Beachten Sie, dass bestimmte betriebssystemspezifische Anweisungen unter Umständen nicht unterstützt werden. Die folgenden Anweisungsfamilien werden in jedem Fall unterstützt:

- Pentium-Familie
- Pentium Pro und Pentium II
- Pentium III
- Pentium 4

Darüber hinaus unterstützt der integrierte Assembler die folgenden Befehlssätze:

- AMD 3DNow! (ab AMD K6 aufwärts)
- AMD Enhanced 3DNow! (ab AMD Athlon aufwärts)

Eine vollständige Beschreibung aller Anweisungen finden Sie in der Dokumentation Ihres Mikroprozessors.

Die Anweisung RET

Der Anweisungs-Opcode RET bewirkt immer eine Near-Rückkehr.

Automatische Sprungoptimierung

Wenn nicht anders angegeben, optimiert der integrierte Assembler Sprunganweisungen durch automatische Auswahl der kürzesten und damit effektivsten Form eines Sprungbefehls. Diese automatische Anpassung wird bei der nicht bedingten Sprunganweisung (JMP) und allen bedingten Sprunganweisungen angewendet, wenn es sich bei dem Ziel um ein Label (und nicht um eine Prozedur oder Funktion) handelt.

Bei einer nicht bedingten Sprunganweisung (JMP) erzeugt der integrierte Assembler einen kurzen Sprung (ein Byte Opcode und ein Byte mit Angabe der Sprungweite), wenn die Adressdifferenz zum Ziel-Label im Bereich von 128 bis 127 Byte liegt. Andernfalls wird ein Near-Sprung generiert (ein Byte Opcode und zwei Byte für die Sprungweite).

Bei einer bedingten Sprunganweisung wird ein kurzer Sprung (ein Byte Opcode und ein Byte mit Angabe der Sprungweite) erzeugt, wenn der Adressabstand zum Ziel-Label im Bereich von 128 bis 127 Byte liegt. Andernfalls generiert der integrierte Assembler einen kurzen Sprung mit der inversen Bedingung, bei dem über einen Near-Sprung zum Ziel-Label gesprungen wird (insgesamt fünf Byte). Beispiel:

JC Stop

In dieser Assembler-Anweisung liegt **Stop** nicht innerhalb der Reichweite eines kurzen Sprungs. Sie wird in folgende Maschinencode-Sequenz umgewandelt:

```
JNC     Skip
JMP     Stop
Skip:
```

Sprünge zu Eintrittspunkten von Prozeduren und Funktionen sind immer Near-Sprünge.

Assembler-Direktiven

Der integrierte Assembler unterstützt drei Definitionsdirektiven: **DB** (Define Byte), **DW** (Define Word) und **DD** (Define Double Word). Jede dieser Direktiven erzeugt Daten, die den durch Kommas voneinander getrennten, nachgestellten Operanden entsprechen.

Direktive	Beschreibung
DB	Define byte: erzeugt eine Byte-Sequenz. Jeder Operand kann ein konstanter Ausdruck mit einem Wert zwischen 128 und 255 oder ein String von beliebiger Länge sein. Konstante Ausdrücke erzeugen Code mit einer Länge von einem Byte. Strings definieren eine Byte-Folge, die den ASCII-Codes der enthaltenen Zeichen entspricht.
DW	Define word: erzeugt eine Word-Sequenz. Jeder Operand kann ein konstanter Ausdruck mit einem Wert zwischen 32.768 und 65.535 oder ein Adressausdruck sein. Bei einem Adressausdruck erzeugt der integrierte Assembler einen Near-Zeiger, d. h. einen Word-Wert mit dem Offset-Anteil der Adresse.
DD	Define double word: erzeugt eine Double-Word-Sequenz. Jeder Operand kann ein konstanter Ausdruck mit einem Wert zwischen 2.147.483.648 und 4.294.967.295 oder ein Adressausdruck sein. Bei einem Adressausdruck erzeugt der integrierte Assembler einen Far-Zeiger, d. h. einen Word-Wert mit dem Offset und einen Word-Wert mit dem Segment der Adresse.
DQ	Define quad word: definiert ein Quad-Word für Int64-Werte.

Die durch die Direktiven **DB**, **DW** und **DD** erzeugten Daten werden wie der Code, der von anderen Anweisungen des integrierten Assemblers erzeugt wird, immer im Code-Segment gespeichert. Zur Erstellung nicht initialisierter Daten im Datensegment müssen Sie die **var**- und **const**-Deklarationen von Delphi verwenden.

Hier einige Beispiele für die Direktiven **DB**, **DW** und **DD**:

```
asm
  DB      FFH           { Ein Byte }
  DB      0,99          { Zwei Byte }
  DB      'A'           { Ord('A') }
  DB      'Hello world...',0DH,0AH { String gefolgt von CR/LF }
  DB      12,'String'   { Delphi-String }
  DW      0FFFFH        { Ein Word }
  DW      0,9999         { Zwei Words }
  DW      'A'           { Identisch mit DB 'A',0 }
  DW      'BA'           { Identisch mit DB 'A','B' }
  DW      MyVar          { Offset von MyVar }
  DW      MyProc         { Offset von MyProc }
  DD      0FFFFFFFFFH  { Ein Double Word }
  DD      0,9999999999  { Zwei Double Words }
  DD      'A'           { Identisch mit DB 'A',0,0,0 }
  DD      'DCBA'         { Identisch mit DB 'A','B','C','D' }
  DD      MyVar          { Zeiger auf MyVar }
  DD      MyProc         { Zeiger auf MyProc }
end;
```

Wenn vor der Direktive **DB**, **DW** oder **DD** ein Bezeichner angegeben wird, führt dies zur Deklaration einer Variablen mit einem Byte, einem Word bzw. einem Double Word an der Speicheradresse der Direktive. Die folgenden Anweisungen sind beispielsweise im Assembler zulässig:

```
ByteVar    DB    ?
WordVar    DW    ?
IntVar     DD    ?
.
.
.
MOV        AL,ByteVar
MOV        BX,WordVar
MOV        ECX,IntVar
```

Der integrierte Assembler unterstützt diese Variablen Deklarationen jedoch nicht. Das einzige Symbol, das in einer Anweisung des integrierten Assemblers definiert werden kann, ist ein Label. Alle Variablen müssen in der Delphi-Syntax deklariert werden. Die obige Konstruktion entspricht folgenden Deklarationen:

```
var
```

```

ByteVar: Byte;
WordVar: Word;
IntVar: Integer;
.

.

.

asm
  MOV AL,ByteVar
  MOV BX,WordVar
  MOV ECX,IntVar
end;

```

Mit den Direktiven **SMALL** und **LARGE** kann die Sprungweite festgelegt werden:

```
MOV EAX, [LARGE $1234]
```

Diese Anweisung generiert eine normale Verschiebung mit einer 32-Bit-Sprungweite (\$00001234).

```
MOV EAX, [SMALL $1234]
```

Die zweite Anweisung generiert eine Verschiebung mit einem Präfix zur Adressgrößenüberschreibung und einer 16-Bit-Sprungweite (\$1234).

Mit der Direktive **SMALL** kann Speicherplatz eingespart werden. Das folgende Beispiel generiert eine Adressgrößenüberschreibung und eine 2-Byte-Adresse (insgesamt 3 Byte):

```
MOV EAX, [SMALL 123]
```

im Gegensatz zu

```
MOV EAX, [123]
```

Letzteres generiert keine Adressgrößenüberschreibung und eine 4-Byte-Adresse (insgesamt also 4 Byte).

Zwei zusätzliche Direktiven ermöglichen dem Assembler-Code den Zugriff auf dynamische und virtuelle Methoden: **VMTOFFSET** und **DMTINDEX**.

VMTOFFSET ruft den Byte-Offset des Tabelleneintrags mit dem virtuellen Methodenzeiger des virtuellen Methodenarguments vom Anfang der virtuellen Methodentabelle (VMT) ab. Diese Direktive benötigt einen vollständig angegebenen Klassennamen mit einem Methodennamen als Parameter (beispielsweise *TExample.VirtualMethod*) oder einen Interface-Namen und einen Interface-Methodennamen.

DMTINDEX ruft den dynamischen Methodentabellenindex der übergebenen dynamischen Methode ab. Diese Direktive benötigt ebenfalls einen vollständig angegebenen Klassennamen mit einem Methodennamen als Parameter, (z. B. *TExample.DynamicMethod*). Sie rufen die dynamische Methode mit *System.@CallDynaInst* mit dem (E)SI-Register auf, das den von **DMTINDEX** abgerufenen Wert enthält.

Anmerkung: Methoden mit der *message*-Direktive werden als dynamische Methoden implementiert und können auch mit **DMTINDEX** aufgerufen werden. Beispiel:

```

TMyClass = class
  procedure x; message MYMESSAGE;
end;

```

Das folgende Beispiel verwendet sowohl **DMTINDEX** als auch **VMTOFFSET** für den Zugriff auf dynamische und virtuelle Methoden:

```

program Project2;
type
  TExample = class
    procedure DynamicMethod; dynamic;
    procedure VirtualMethod; virtual;
  end;

```

```

procedure TExample.DynamicMethod;
begin
end;

procedure TExample.VirtualMethod;
begin
end;

procedure CallDynamicMethod(e: TExample);
asm
  // Speichern des ESI-Registers
  PUSH ESI
  // Instanzzeiger muss in EAX vorliegen
  MOV EAX, e
  // DMT-Eintragsindex muss in (E)SI vorliegen
  MOV ESI, DMTINDEX TExample.DynamicMethod
  // Aufruf der Methode
  CALL System.@CallDynaInst
  // ESI-Register wiederherstellen
  POP ESI
end;

procedure CallVirtualMethod(e: TExample);
asm
  // Instanzzeiger muss in EAX vorliegen
  MOV EAX, e
  // MT-Tabelleneintrag abrufen
  MOV EDX, [EAX]
  // Aufruf der Methode bei Offset VMTOFFSET
  CALL DWORD PTR [EDX + VMTOFFSET TExample.VirtualMethod]
end;

var
  e: TExample;
begin
  e := TExample.Create;
  try
    CallDynamicMethod(e);
    CallVirtualMethod(e);
  finally
    e.Free;
  end;
end.

```

Operanden

Die Operanden des integrierten Assemblers sind Ausdrücke, die aus Konstanten, Registern, Symbolen und Operatoren bestehen.

Die folgenden reservierten Wörter haben bei ihrer Verwendung in Operanden eine vordefinierte Bedeutung:

Reservierte Wörter im integrierten Assembler

AH	CL	DX	ESP	mm4	SHL	WORD
AL	CS	EAX	FS	mm5	SHR	xmm0

AND	CX	EBP	GS	mm6	SI	xmm1
AX	DH	EBX	HIGH	mm7	SMALL	xmm2
BH	DI	ECX	LARGE	MOD	SP	xmm3
BL	DL	EDI	LOW	NOT	SS	xmm4
BP	CL	EDX	mm0	OFFSET	ST	xmm5
BX	DMTINDEX	EIP	mm1	OR	TBYTE	xmm6
BYTE	DS	ES	mm2	PTR	TYPE	xmm7
CH	DWORD	ESI	mm3	QWORD	VMTOFFSET	XOR

Reservierte Wörter haben immer Vorrang vor benutzerdefinierten Bezeichnern. Beispiel:

```
var
  Ch: Char;
  .
  .
  .
asm
  MOV    CH, 1
end;
```

Im vorangegangenen Code-Fragment wird 1 nicht in die Variable *Ch*, sondern in das Register CH geladen. Wenn Sie auf ein benutzerdefiniertes Symbol zugreifen möchten, das den Namen eines reservierten Wortes trägt, müssen Sie den Operator **&** zum Überschreiben des Bezeichners verwenden:

```
MOV&Ch, 1
```

Benutzerdefinierte Bezeichner sollten möglichst nie mit den Namen reservierter Wörter belegt werden.

Siehe auch

Verwendung des integrierten Assemblers (siehe Seite 1034)

Assembler-Ausdrücke (siehe Seite 1040)

Assembler-Prozeduren und -Funktionen (siehe Seite 1048)

4.1.1.12.3 Assembler-Ausdrücke (nur Win32)

Der integrierte Assembler wertet alle Ausdrücke als 32-Bit-Integer aus. Gleitkomma- und String-Werte werden mit Ausnahme von String-Konstanten nicht unterstützt. Der integrierte Assembler ist nur im Win32-Compiler von Delphi verfügbar.

Ausdrücke werden aus Ausdruckselementen und Operatoren erstellt und gehören zu einer bestimmten Ausdrucksklasse und zu einem bestimmten Ausdruckstyp. In diesem Thema werden folgende Bereiche erläutert:

- Unterschiede zwischen Ausdrücken in Delphi und Assembler
- Ausdruckselemente
- Ausdrucksklassen
- Ausdruckstypen
- Ausdrucksoperatoren

Unterschiede zwischen Ausdrücken in Delphi und Assembler

Der größte Unterschied zwischen Delphi-Ausdrücken und Ausdrücken des integrierten Assemblers besteht darin, dass Assembler-Ausdrücke einen konstanten Wert ergeben müssen, d. h. einen Wert, der während der Compilierung berechnet werden kann. Ausgehend von den Deklarationen

```
const
  X = 10;
  Y = 20;
var
  Z: Integer;
```

ist folgende Assembler-Anweisung zulässig:

```
asm
  MOV      Z, X+Y
end;
```

Da `X` und `Y` Konstanten sind, ist der Ausdruck `X + Y` nur eine andere Möglichkeit zur Darstellung der Konstante 30. Die resultierende Anweisung bewirkt eine direkte Speicherung des Wertes 30 in der Variablen `Z`. Wenn `X` und `Y` aber Variablen sind, also

```
var
  X, Y: Integer;
```

kann der integrierte Assembler den Wert von `X + Y` nicht während der Compilierung berechnen. In diesem Fall müssten Sie folgende Anweisung verwenden, um die Summe von `X` und `Y` in `Z` zu speichern:

```
asm
  MOV      EAX, X
  ADD      EAX, Y
  MOV      Z, EAX
end;
```

In einem Delphi-Ausdruck zeigt eine Variablenreferenz auf den *Inhalt* der Variable. In einem Assembler-Ausdruck hingegen gibt eine Variablenreferenz die *Adresse* der Variable an. Beispielsweise bezieht sich in Delphi der Ausdruck `X + 4`, in dem `X` eine Variable ist, auf den Inhalt von `X + 4`. Im integrierten Assembler bedeutet dieser Ausdruck, dass sich der Inhalt des Word an einer Adresse befindet, die um vier Byte höher ist als die Adresse von `X`. So ist zwar folgende Anweisung zulässig:

```
asm
  MOV      EAX, X+4
end;
```

Der Wert von `X + 4` wird aber nicht in `AX` geladen, sondern der Word-Wert, der vier Bytes hinter `X` gespeichert ist. Um 4 zum Inhalt von `X` zu addieren, müssen Sie folgende Anweisung verwenden:

```
asm
  MOV      EAX, X
  ADD      EAX, 4
end;
```

Ausdruckselemente

Zu den Elementen eines Ausdrucks gehören Konstanten, Register und Symbole.

Numerische Konstanten

Bei numerischen Konstanten muss es sich um Integer-Werte zwischen 2.147.483.648 und 4.294.967.295 handeln.

Per Voreinstellung wird bei numerischen Konstanten die dezimale Schreibweise verwendet. Der integrierte Assembler unterstützt aber auch die binäre, oktale und hexadezimale Notation. Sie können die binäre Schreibweise verwenden, indem Sie nach der Zahl den Buchstaben `B` angeben. Bei der oktalen Notation folgt nach der Zahl der Buchstabe `O`. Um die hexadezimale Schreibweise zu verwenden, geben Sie nach der Zahl den Buchstaben `H` oder vor dem Wert das Zeichen `$` ein.

Numerische Konstanten müssen mit den Ziffern 0 bis 9 oder mit dem Zeichen `$` beginnen. Wenn Sie eine hexadezimale Konstante mit dem Suffix `H` angeben und die erste signifikante Stelle eine hexadezimale Ziffer zwischen A und F ist, müssen Sie eine zusätzliche Null an den Beginn der Zahl stellen. So handelt es sich beispielsweise bei `0BAD4H` und `$BAD4` um hexadezimale Konstanten, bei `BAD4H` aber um einen Bezeichner, weil der Ausdruck mit einem Buchstaben beginnt.

String-Konstanten

String-Konstanten müssen in halbe oder ganze Anführungszeichen eingeschlossen werden. Zwei aufeinander folgende Anführungszeichen desselben Typs wie die umgebenden Anführungszeichen werden als ein Zeichen interpretiert. Es folgen einige Beispiele für String-Konstanten:

```
'Z'  
'Delphi'  
'Linux'  
"Das war's dann"  
'"Das war's dann," sagte er.'  
'100'  
'''  
'''
```

In **DB**-Direktiven sind String-Konstanten von beliebiger Länge erlaubt. Sie bewirken die Zuweisung einer Byte-Folge mit den ASCII-Werten der Zeichen. In allen anderen Fällen darf eine String-Konstante nicht länger als vier Zeichen sein und muss einen numerischen Wert angeben, der in einem Ausdruck zulässig ist. Der numerische Wert einer String-Konstante wird wie folgt berechnet:

$Ord(Ch1) + Ord(Ch2) \text{ shl } 8 + Ord(Ch3) \text{ shl } 16 + Ord(Ch4) \text{ shl } 24$

Dabei ist **Ch1** das am weitesten rechts stehende (letzte) Zeichen und **Ch4** das am weitesten links stehende (erste) Zeichen. Wenn der String weniger als vier Zeichen enthält, werden die am weitesten links stehenden Zeichen als Nullen interpretiert. Die folgende Tabelle enthält einige Beispiele für String-Konstanten und die entsprechenden numerischen Werte:

Beispiele für String-Konstanten und ihre Werte

String	Wert
'a'	00000061H
'ba'	00006261H
'cba'	00636261H
'dcba'	64636261H
' a'	00006120H
' a'	20202061H
'a' * 2	000000E2H
'a'-'A'	00000020H
not 'a'	FFFFFFFFFF9EH

Register

Die reservierten Symbole in der folgenden Tabelle geben die CPU-Register im integrierten Assembler an:

CPU-Register

32-Bit-Allzweckregister	EAX EBX ECX EDX	32-Bit-Zeiger- oder -Indexregister	ESP EBP ESI EDI
16-Bit-Allzweckregister	AX BX CX DX	16-Bit-Zeiger- oder -Indexregister	SP BP SI DI
Untere 8-Bit-Register	AL BL CL DL	16-Bit-Segmentregister	CS DS SS ES
		32-Bit-Segmentregister	FS GS
Obere 8-Bit-Register	AH BH CH DH	Coprozessor-Registerstapel	ST

Wenn ein Operand nur aus einem Registernamen besteht, wird er als Register-Operand bezeichnet. Als Register-Operanden können alle Register verwendet werden. Einige Register lassen sich auch in einem anderen Kontext einsetzen.

Die Basisregister (BX und BP) und die Indexregister (SI und DI) können zur Kennzeichnung der Indizierung in eckigen Klammern angegeben werden. Folgende Kombinationen von Basis-/Indexregistern sind zulässig: [BX], [BP], [SI], [DI], [BX+SI], [BX+DI], [BP+SI] und [BP+DI]. Sie können auch mit allen 32-Bit-Registern indizieren, z. B. [EAX+ECX], [ESP] und [ESP+EAX+5].

Die Segmentregister (ES, CS, SS, DS, FS und GS) werden unterstützt, aber in 32-Bit-Anwendungen ist die Verwendung von Segmenten normalerweise nicht sinnvoll.

Das Symbol ST bezeichnet das oberste Register im 8087-Gleitkommaregister-Stack. Jedes der acht Gleitkommaregister kann mit ST(X) referenziert werden, wobei X eine Konstante von 0 bis 7 ist, die den Abstand vom oberen Stack-Ende angibt.

Symbole

Der integrierte Assembler ermöglicht den Zugriff auf nahezu alle Delphi-Bezeichner in Assembler-Ausdrücken, einschließlich Konstanten, Typen, Variablen, Prozeduren und Funktionen. Außerdem ist im integrierten Assembler das spezielle Symbol **@Result** implementiert, das der Variablen *Result* im Anweisungsteil einer Funktion entspricht. So kann beispielsweise die Funktion

```
function Sum(X, Y: Integer): Integer;
begin
  Result := X + Y;
end;
```

in Assembler folgendermaßen angegeben werden:

```
function Sum(X, Y: Integer): Integer; stdcall;
begin
  asm
    MOV    EAX, X
    ADD    EAX, Y
    MOV    @Result, EAX
  end;
end;
```

Die folgenden Symbole dürfen in **asm**-Anweisungen nicht verwendet werden:

- Standardprozeduren und -funktionen (z. B. **WriteLn** und **Chr**).
- String-, Gleitkomma- und Mengenkonstanten (außer beim Laden von Registern).
- Label, die nicht im aktuellen Block deklariert sind.
- Das Symbol **@Result** außerhalb einer Funktion.

Die folgende Tabelle enthält die Symbolarten, die in **asm**-Anweisungen verwendet werden können.

Im integrierten Assembler verwendbare Symbole

Symbol	Wert	Klasse	Typ
Label	Adresse des Labels	Speicherreferenz	Größe des Typs
Konstante	Wert der Konstante	Direkter Wert	0
Typ	0	Speicherreferenz	Größe des Typs
Feld	Offset des Feldes	Speicher	Größe des Typs
Variable	Adresse der Variable oder Adresse eines Zeigers auf die Variable	Speicherreferenz	Größe des Typs
Prozedur	Adresse der Prozedur	Speicherreferenz	Größe des Typs
Funktion	Adresse der Funktion	Speicherreferenz	Größe des Typs
Unit	0	Direkter Wert	0

@Result	Ergebnisvariablen-Offset	Speicherreferenz	Größe des Typs
---------	--------------------------	------------------	----------------

Bei deaktivierter Optimierung werden lokale (also in Prozeduren und Funktionen deklarierte) Variablen immer auf dem Stack abgelegt. Der Zugriff erfolgt immer relativ zu EBP. Der Wert eines lokalen VariablenSymbols besteht in seinem mit Vorzeichen versehenen Offset von EBP. Der Assembler addiert zu Referenzen auf lokale Variablen automatisch [EBP] hinzu. Betrachten Sie beispielsweise die folgende Deklaration:

var Count: Integer;

enthält, wird die Anweisung

MOV EAX, Count

in MOV EAX, [EBP4] assembliert.

Der integrierte Assembler behandelt einen **var**-Parameter immer als 32-Bit-Zeiger. Die Größe eines **var**-Parameters beträgt immer 4 Byte. Die Syntax für den Zugriff auf einen **var**-Parameter unterscheidet sich von derjenigen für einen Wertparameter. Für den Zugriff auf den Inhalt eines **var**-Parameters müssen Sie zuerst den 32-Bit-Zeiger laden und dann auf die Speicheradresse zugreifen, auf die er zeigt. Zum Beispiel:

```
function Sum(var X, Y: Integer): Integer; stdcall;
begin
  asm
    MOV EAX, X
    MOV EDX, [EAX]
    MOV EAX, Y
    ADD EAX, [EDX]
    MOV @Result, EAX
  end;
end;
```

Bezeichner können in **asm**-Anweisungen qualifiziert werden. Ausgehend von den Deklarationen

```
type
  TPoint = record
    X, Y: Integer;
  end;
  TRect = record
    A, B: TPoint;
  end;
var
  P: TPoint;
  R : TRect;
```

kann mit folgenden Konstruktionen in einer **asm**-Anweisung auf die Felder zugegriffen werden:

```
MOV EAX, P.X
MOV EDX, P.Y
MOV ECX, R.A.X
MOV EBX, R.B.Y
```

Typbezeichner können zur einfachen und schnellen Erstellung von Variablen verwendet werden. Die folgenden Anweisungen erzeugen denselben Maschinencode, der den Inhalt von [EDX] in das Register EAX lädt:

```
MOV EAX, (TRect PTR [EDX]).B.X
MOV EAX, TRect([EDX]).B.X
MOV EAX, TRect[EDX].B.X
MOV EAX, [EDX].TRect.B.X
```

Ausdrucksklassen

Der integrierte Assembler unterteilt Ausdrücke in drei Klassen: Register, Speicherreferenzen und direkte Werte.

Ausdrücke, die nur aus einem Registernamen bestehen, nennt man Registerausdrücke. Beispiele hierfür sind AX, CL, DI und ES. Registerausdrücke, die als Operanden verwendet werden, veranlassen den Assembler zur Erzeugung von Anweisungen,

die auf die CPU-Register zugreifen.

Ausdrücke, die Speicheradressen bezeichnen, nennt man Speicherreferenzen. Zu dieser Kategorie gehören Label, Variablen, typisierte Konstanten, Prozeduren und Funktionen von Delphi.

Ausdrücke, bei denen es sich nicht um Register handelt und die auch nicht auf Speicheradressen zeigen, werden als direkte Werte bezeichnet. Zu dieser Gruppe gehören untypisierte Konstanten und Typbezeichner von Delphi.

Wenn direkte Werte und Speicherreferenzen als Operanden verwendet werden, führt dies zu unterschiedlichem Code. Zum Beispiel:

```
const
  Start = 10;
var
  Count: Integer;
  .
  .
  .
asm
  MOV      EAX,Start    { MOV EAX,xxxx }
  MOV      EBX,Count    { MOV EBX,[xxxx] }
  MOV      ECX,[Start]   { MOV ECX,[xxxx] }
  MOV      EDX,OFFSET Count { MOV EDX,xxxx }
end;
```

Da `Start` ein direkter Wert ist, wird das erste `MOV` in eine Move-Immediate-Anweisung assembliert. Das zweite `MOV` wird in eine Move-Memory-Anweisung übersetzt, weil `Count` eine Speicherreferenz ist. Im dritten `MOV` wird `Start` wegen der eckigen Klammern in eine Speicherreferenz umgewandelt (in diesem Fall handelt es sich um das Word mit dem Offset 10 im Datensegment). Im vierten `MOV` sorgt der Operator `OFFSET` für die Konvertierung von `Count` in einen direkten Wert (mit dem Offset von `Count` im Datensegment).

Die eckigen Klammern und der Operator `OFFSET` ergänzen einander. Die folgende `asm`-Anweisung erzeugt denselben Maschinencode wie die ersten beiden Zeilen der obigen `asm`-Anweisung:

```
asm
  MOV      EAX,OFFSET [Start]
  MOV      EBX,[OFFSET Count]
end;
```

Bei Speicherreferenzen und direkten Werten findet eine weitere Unterteilung in verschiebbare und absolute Ausdrücke statt. Unter einer Verschiebung versteht man den Vorgang, bei dem der Linker Symbolen eine absolute Adresse zuweist. Ein verschiebbarer Ausdruck bezeichnet einen Wert, für den beim Linken eine Verschiebung (Relokation) erforderlich ist. Dagegen bezeichnet ein absoluter Ausdruck einen Wert, bei dem dies nicht nötig ist. In der Regel handelt es sich bei Ausdrücken, die ein Label, eine Variable, eine Prozedur oder eine Funktion referenzieren, um verschiebbare Ausdrücke, weil die endgültige Adresse dieser Symbole zur Compilierungszeit nicht bekannt ist. Absolut sind dagegen Ausdrücke, die ausschließlich Konstanten bezeichnen.

Im integrierten Assembler kann mit absoluten Werten jede Operation ausgeführt werden. Mit verschiebbaren Ausdrücken ist dagegen nur die Addition und Subtraktion von Konstanten möglich.

Ausdruckstypen

Jedem Assembler-Ausdruck ist ein bestimmter Typ (genauer gesagt eine bestimmte Größe) zugeordnet, weil der Assembler den Typ eines Ausdrucks einfach aus der Größe seiner Speicherposition abliest. Beispielsweise hat eine Integer-Variable den Typ `Vier`, weil sie vier Byte Speicherplatz belegt. Der integrierte Assembler führt, wenn möglich, immer eine Typprüfung durch. Ein Beispiel:

```
var
  QuitFlag: Boolean;
  OutBufPtr: Word;
  .
  .
```

```

asm
MOV      AL,QuitFlag
MOV      BX,OutBufPtr
end;

```

Der Assembler prüft, ob die Größe von `QuitFlag` eins (ein Byte) und die Größe von `OutBufPtr` zwei (ein Word) beträgt. Die folgende Anweisung führt zu einem Fehler:

```
MOV      DL,OutBufPtr
```

Das Problem liegt darin, dass `DL` nur ein Byte groß ist, während `OutBufPtr` ein Word ist. Der Typ einer Speicherreferenz kann durch eine Typumwandlung geändert werden. Die obige Anweisung müsste also folgendermaßen formuliert werden:

```

MOV      DL,BYTE PTR OutBufPtr
MOV      DL,Byte(OutBufPtr)
MOV      DL,OutBufPtr.Byte

```

Diese `MOV`-Anweisungen referenzieren das erste (niederwertige) Byte der Variablen `OutBufPtr`.

Es gibt Fälle, in denen eine Speicherreferenz untypisiert ist. Ein Beispiel hierfür ist ein direkter Wert (`Buffer`), der in eckige Klammern gesetzt ist:

```

procedure Example(var Buffer);
asm
  MOV AL,      [Buffer]
  MOV CX,      [Buffer]
  MOV EDX,     [Buffer]
end;

```

Der integrierte Assembler lässt beide Anweisungen zu, da der Anweisung `[Buffer]` kein Typ zugeordnet ist. `[Buffer]` bezeichnet einfach den Inhalt der von `Buffer` angegebenen Adresse und der Typ kann anhand des ersten Operanden festgestellt werden (Byte für `AL`, Word für `CX` und Double Word für `EDX`).

Falls sich der Typ nicht über einen anderen Operanden ermitteln lässt, verlangt der integrierte Assembler eine explizite Typumwandlung. Zum Beispiel:

```
INC      BYTE PTR [ECX]
IMUL     WORD PTR [EDX]
```

Die folgende Tabelle enthält die vordefinierten Typensymbole, die der integrierte Assembler zusätzlich zu den aktuell deklarierten Delphi-Typen bereitstellt.

Vordefinierte Typensymbole

Symbol	Typ
BYTE	1
WORD	2
DWORD	4
QWORD	8
TBYTE	10

Ausdrucksoperatoren

Der integrierte Assembler stellt eine Vielzahl von Operatoren bereit. Die Regeln für die Rangfolge der Operatoren unterscheiden sich von den Regeln in Delphi. Beispielsweise haben die Operatoren für Addition und Subtraktion in einer `asm`-Anweisung Vorrang gegenüber `AND`. Die folgende Tabelle enthält die Ausdrucksoperatoren des integrierten Assemblers. Die Operatoren sind nach ihrer Rangfolge sortiert.

Rangfolge der Ausdrucksoperatoren des integrierten Assemblers

Operatoren	Bemerkungen	Rangfolge
&		Höchste Stufe
(...), [...],., HIGH, LOW		
+, -	Unäres + und -	
:		
OFFSET, TYPE, PTR, *, /, MOD, SHL, SHR, +, -	Binäres + und -	
NOT, AND, OR, XOR		Niedrigste Stufe

Die folgende Tabelle fasst die Ausdrucksoperatoren des integrierten Assemblers zusammen.

Erläuterung der Ausdrucksoperatoren des integrierten Assemblers

Operator	Beschreibung
&	Überschreiben von Bezeichnern. Der Bezeichner, der unmittelbar auf das Zeichen & folgt, wird als benutzerdefiniertes Symbol betrachtet. Dies gilt auch dann, wenn er mit einem reservierten Symbol des integrierten Assemblers identisch ist.
(...)	Unterausdruck. Ausdrücke in Klammern werden vollständig ausgewertet und als einzelnes Ausdruckselement betrachtet. Optional kann dem Ausdruck in Klammern ein weiterer Ausdruck vorangestellt werden. Das Resultat ist in diesem Fall die Summe der Werte der beiden Ausdrücke. Der Typ des ersten Ausdrucks bestimmt den Ergebnistyp.
[...]	Speicherreferenz. Der Ausdruck in eckigen Klammern wird vollständig ausgewertet und dann als einzelnes Ausdruckselement betrachtet. Das Resultat ist in diesem Fall die Summe der Werte der beiden Ausdrücke. Der Typ des ersten Ausdrucks bestimmt den Ergebnistyp. Das Ergebnis ist immer eine Speicherreferenz.
.	Selektor für Strukturelemente. Das Resultat ergibt sich aus der Addition der Ausdrücke vor und nach dem Punkt. Der Typ des Ausdrucks nach dem Punkt bestimmt den Ergebnistyp. Im Ausdruck nach dem Punkt kann auf Symbole, die zum Gültigkeitsbereich des Ausdrucks vor dem Punkt gehören, zugegriffen werden.
HIGH	Gibt die höherwertigen acht Bits des Word-Ausdrucks zurück, der auf den Operator folgt. Der Ausdruck muss ein absoluter direkter Wert sein.
LOW	Gibt die niederwertigen acht Bits des Word-Ausdrucks zurück, der auf den Operator folgt. Der Ausdruck muss ein absoluter direkter Wert sein.
+	Unäres Plus. Liefert den auf das Pluszeichen folgenden Ausdruck ohne Änderungen zurück. Der Ausdruck muss ein absoluter direkter Wert sein.
-	Unäres Minus. Liefert den negativen Wert des Ausdrucks zurück, der auf das Minuszeichen folgt. Der Ausdruck muss ein absoluter direkter Wert sein.
+	Addition. Die Ausdrücke können direkte Werte oder Speicherreferenzen sein. Nur einer der Ausdrücke darf aus einem verschiebbaren Wert bestehen. Handelt es sich bei einem der Ausdrücke um einen verschiebbaren Wert, ist das Ergebnis ebenfalls ein verschiebbarer Wert. Ist einer der Ausdrücke eine Speicherreferenz, ist auch das Ergebnis eine Speicherreferenz.
-	Subtraktion. Der erste Ausdruck kann zu einer beliebigen Klasse gehören, der zweite muss ein absoluter direkter Wert sein. Das Ergebnis gehört zur gleichen Klasse wie der erste Ausdruck.

4

:	Überschreiben von Segmenten. Teilt dem Assembler mit, dass der Ausdruck nach dem Doppelpunkt zu dem Segment gehört, das über den Segmentregisternamen (CS, DS, SS, FS, GS oder ES) vor dem Doppelpunkt angegeben ist. Das Ergebnis ist eine Speicherreferenz mit dem Wert des Ausdrucks nach dem Doppelpunkt. Wenn in einem Anweisungsoperanden das Überschreiben eines Segments verwendet wird, wird der Anweisung eine entsprechende Präfixanweisung zur Segmentüberschreibung vorangestellt. Dies stellt sicher, dass das angegebene Segment ausgewählt ist.
OFFSET	Liefert den Offset-Anteil (Double Word) des Ausdrucks zurück, der auf den Operator folgt. Das Ergebnis ist ein direkter Wert.
TYPE	Liefert den Typ (die Größe in Byte) des Ausdrucks zurück, der auf den Operator folgt. Der Typ eines direkten Wertes ist 0.
PTR	Typumwandlungsoperator. Das Ergebnis ist eine Speicherreferenz mit dem Wert des Ausdrucks, der auf den Operator folgt, und mit dem Typ des Ausdrucks, der dem Operator vorangeht.
*	Multiplikation. Beide Ausdrücke müssen absolute, direkte Werte sein. Das Ergebnis ist ebenfalls ein absoluter direkter Wert.
/	Integerdivision. Beide Ausdrücke müssen absolute, direkte Werte sein. Das Ergebnis ist ebenfalls ein absoluter direkter Wert.
MOD	Rest einer Integerdivision. Beide Ausdrücke müssen absolute, direkte Werte sein. Das Ergebnis ist ebenfalls ein absoluter direkter Wert.
SHL	Logische Linksverschiebung. Beide Ausdrücke müssen absolute, direkte Werte sein. Das Ergebnis ist ebenfalls ein absoluter direkter Wert.
SHR	Logische Rechtsverschiebung. Beide Ausdrücke müssen absolute, direkte Werte sein. Das Ergebnis ist ebenfalls ein absoluter direkter Wert.
NOT	Bitweise Negation. Der Ausdruck muss ein absoluter, direkter Wert sein. Das Ergebnis ist ebenfalls ein absoluter direkter Wert.
AND	Bitweises AND. Beide Ausdrücke müssen absolute, direkte Werte sein. Das Ergebnis ist ebenfalls ein absoluter direkter Wert.
OR	Bitweises OR. Beide Ausdrücke müssen absolute, direkte Werte sein. Das Ergebnis ist ebenfalls ein absoluter direkter Wert.
XOR	Bitweises exklusives OR. Beide Ausdrücke müssen absolute, direkte Werte sein. Das Ergebnis ist ebenfalls ein absoluter direkter Wert.

Siehe auch

Verwendung des integrierten Assemblers (siehe Seite 1034)

Assembler-Syntax (siehe Seite 1035)

Assembler-Prozeduren und -Funktionen (siehe Seite 1048)

4.1.1.12.4 Assembler-Prozeduren und -Funktionen (nur Win32)

Mit dem integrierten Assembler können Sie komplette Prozeduren und Funktionen schreiben, für die die keine begin...end-Anweisung erforderlich ist. Dieses Thema enthält Informationen zu folgenden Bereichen:

- Compiler-Optimierungen
- Funktionsergebnisse

Der integrierte Assembler ist nur im Win32-Compiler von Delphi verfügbar.

Compiler-Optimierungen

Im Folgenden sehen Sie ein Beispiel für eine gültige Funktion:

```
function LongMul(X, Y: Integer): Longint;
asm
  MOV     EAX, X
  IMUL  Y
end;
```

Der Compiler führt für diese Routinen verschiedene Optimierungen durch:

- Der Compiler erstellt keinen Code zum Kopieren von Wert-Parametern in lokale Variablen. Dies betrifft alle Wert-Parameter vom Typ String und alle anderen Wert-Parameter, deren Größe nicht ein, zwei oder vier Byte beträgt. Innerhalb der Routine müssen derartige Parameter als **var**-Parameter behandelt werden.
- Der Compiler weist keine Funktionsergebnis-Variable zu, und eine Referenz auf das Symbol **@Result** ist ein Fehler. Eine Ausnahme bilden Funktionen, die eine Referenz auf einen String, eine Variante oder ein Interface zurückliefern. Die aufrufende Routine weist diesen Typen immer einen **@Result**-Zeiger zu.
- Der Compiler generiert nur Stackframes für verschachtelte Routinen, für Routinen mit lokalen Parametern oder für Routinen, die über Parameter im Stack verfügen.
- Locals** ist die Größe der lokalen Variablen, **Params** die Größe der Parameter. Wenn sowohl **Locals** als auch **Params** Null ist, existiert kein Eintrittscode, und der Austrittscode besteht nur aus einer RET-Anweisung.

Der automatisch erzeugte Eintritts- und Austrittscode für Routinen sieht folgendermaßen aus:

```
PUSH      EBP      ;Vorhanden, wenn Locals <> 0 oder Params <> 0
MOV       EBP,ESP   ;Vorhanden, wenn Locals <> 0 oder Params <> 0
SUB       ESP,Locals ;Vorhanden, wenn Locals <> 0
.
.
.
MOV       ESP,EBP   ;Vorhanden, wenn Locals <> 0
POP       EBP      ;Vorhanden, wenn Locals <> 0 oder Params <> 0
RET       Params    ;Immer vorhanden
```

Wenn lokale Variablen Varianten, lange Strings oder Interfaces enthalten, werden sie mit Null initialisiert, aber nach der Verarbeitung nicht finalisiert.

Funktionsergebnisse

Assembler-Funktionen liefern ihre Ergebnisse folgendermaßen zurück:

- Ordinale Werte werden in AL (8-Bit-Werte), AX (16-Bit-Werte) oder EAX (32-Bit- Werte) zurückgeliefert.
- Reelle Werte werden in ST(0) über den Register-Stack des Coprozessors zurückgegeben. (**Currency**-Werte werden mit dem Faktor 10000 skaliert.)
- Zeiger einschließlich langer Strings werden in EAX zurückgeliefert.
- Kurze Strings und Varianten werden an die temporären Adresse zurückgegeben, auf die **@Result** zeigt.

Siehe auch

Verwendung des integrierten Assemblers (siehe Seite 1034)

Assembler-Syntax (siehe Seite 1035)

Assembler-Ausdrücke (siehe Seite 1040)

4.1.1.13 Benutzerdefinierte Attribute in .NET

.NET Framework-Assemblierungen sind selbstbeschreibende Entitäten. Sie enthalten Quelltext in einer Zwischensprache, der beim Laden der Assemblierung in den nativen Maschinencode übersetzt wird. Darüber hinaus enthalten Assemblierungen zahlreiche Informationen über den Quelltext. Der Compiler gibt diese so genannten Metadaten während der Verarbeitung des Quelltexts in die Assemblierung aus. In anderen Programmierumgebungen kann nach dem Compilieren des Quelltexts nicht mehr auf die Metadaten zugegriffen werden, da diese Informationen während des Compilervorgangs verloren gehen. Auf der

.NET-Plattform besteht jedoch die Möglichkeit eines Zugriffs auf Metadaten mittels Laufzeit-Reflection-Diensten.

Das .NET Framework ermöglicht es, die vom Compiler ausgegebenen Metadaten um eigene beschreibende Attribute zu erweitern. Diese benutzerdefinierten Attribute entsprechen in etwa den Schlüsselwörtern einer Programmiersprache und werden zusammen mit anderen Metadaten in der Assemblierung gespeichert.

- Benutzerdefinierte Attribute deklarieren
- Benutzerdefinierte Attribute verwenden
- Benutzerdefinierte Attribute und Interfaces

Benutzerdefinierte Attribute deklarieren

Ein benutzerdefiniertes Attribut wird auf die gleiche Weise wie eine Klasse erstellt. Eine benutzerdefinierte Attributklasse verfügt über einen Konstruktor sowie über Eigenschaften für das Zuweisen und Abrufen von Statusdaten zum Attribut. Benutzerdefinierte Attribute müssen von TCustomAttribute abgeleitet werden. Im folgenden Beispiel wird ein benutzerdefiniertes Attribut mit einem Konstruktor und zwei Eigenschaften deklariert:

```
type
  TCustomCodeAttribute = class(TCustomAttribute)
  private
    Fprop1 : integer;
    Fprop2 : integer;
    aVal   : integer;
    procedure Setprop1(p1 : integer);
    procedure Setprop2(p2 : integer);
  public
    constructor Create(const myVal : integer);
    property prop1 : integer read Fprop1 write Setprop1;
    property prop2 : integer read Fprop2 write Setprop2;
  end;
```

Die Implementierung des Konstruktors könnte folgendermaßen aussehen:

```
constructor TCustomCodeAttribute.Create(const myVal : integer);
begin
  inherited Create;
  aVal := myVal;
end;
```

Delphi für .NET unterstützt die Erstellung benutzerdefinierter Attributklassen (siehe oben) sowie alle vom .NET Framework bereitgestellten benutzerdefinierten Attribute.

Benutzerdefinierte Attribute verwenden

Ein benutzerdefiniertes Attribut wird unmittelbar vor der Quelltexteinheit eingefügt, auf die es angewendet wird. Sie können Attribute vor folgenden Elementen einfügen:

- Variablen und Konstanten
- Prozeduren und Funktionen
- Funktionsergebnisse
- Prozedur- und Funktionsparameter
- Typen
- Felder, Eigenschaften und Methoden

Delphi für .NET unterstützt die Verwendung von benannten Eigenschaften bei der Initialisierung. Dabei kann es sich um die Namen von Eigenschaften, von **public**-Feldern oder den Namen der benutzerdefinierte Attributklasse handeln. Benannte Eigenschaften folgen auf die anderen Parameter, die an den Konstruktor übergeben werden müssen. Beispiel:

```
[TCustomCodeAttribute(1024, prop1=512, prop2=128)]
TMyClass = class(TObject)
  ...

```

```
end;
```

Diese Anweisung wendet das oben deklarierte benutzerdefinierte Attribut auf die Klasse `TMyClass` an.

Der erste Parameter (1024) ist der vom Konstruktor benötigte Wert. Darauf folgen zwei Eigenschaften, die im benutzerdefinierte Attribut definiert sind.

Wenn Sie ein benutzerdefiniertes Attribut vor einer Liste mit Variablen Deklarationen angeben, wird das Attribut auf alle Variablen in der Liste angewendet. Beispiel:

```
var
  [TCustomAttribute(1024, prop1=512, prop2=128)]
  x, y, z: Integer;
```

Aufgrund dieser Anweisung wird `TCustomAttribute` auf die drei Variablen `x`, `y` und `z` angewendet.

Auf Typen angewendete benutzerdefinierte Attribute können zur Laufzeit mit der Methode `GetCustomAttributes` der Klasse `Type` ermittelt werden. Der folgende Delphi-Quelltext veranschaulicht die Abfrage von benutzerdefinierten Attributen zur Laufzeit:

```
var
  F: TMyClass;           // TMyClass ist bereits deklariert
  T: System.Type;
  A: array of TObject; // Dient zur Aufnahme der benutzerdefinierten Attribute
  I: Integer;

begin
  F := TMyClass.Create;
  T := F.GetType;
  A := T.GetCustomAttributes(True);

  // Ausgabe des Typnamens, dann Schleife über die benutzerdefinierten
  // Attribute, die folgender Aufruf liefert
  // System.Type.GetCustomAttributes.
  Writeln(T.FullName);
  for I := Low(A) to High(A) do
    Writeln(A[I].GetType.FullName);
end.
```

Das benutzerdefinierte Attribut `DllImport`

Sie können nicht verwaltete Win32-APIs (und anderen nicht verwalteten Code) aufrufen, indem Sie der Funktionsdeklaration das benutzerdefinierte Attribut `DllImport` voranstellen. Dieses Attribut befindet sich im Namespace `System.Runtime.InteropServices`. Ein Beispiel:

```
Program HelloWorld2;

  // Bei Verwendung des Attributs DllImport muss unbedingt die Unit InteropServices
  // angegeben werden
  uses System.Runtime.InteropServices;

  [DllImport('user32.dll')]
  function MessageBeep(uType : LongWord) : Boolean; external;

begin
  MessageBeep(LongWord(-1));
end.
```

Das Schlüsselwort `external` muss angegeben werden, um den Block in der Funktionsdeklaration zu ersetzen. Alle anderen Attribute, etwa die Aufrufkonvention, können über das benutzerdefinierte Attribut `DllImport` übergeben werden.

Benutzerdefinierte Attribute und Interfaces

In Delphi muss die GUID (falls vorhanden) unmittelbar auf die Deklaration eines Interface folgen. Da für GUIDs und für benutzerdefinierte Attribute eine ähnlich Syntax verwendet wird, muss der Compiler zwischen diesen beiden Entitäten unterscheiden können (benutzerdefinierte Attribute beziehen sich auf die nächste Deklaration, GUID-Bezeichner auf die vorherige). Andernfalls würde der Compiler versuchen, ein Attribut auf das erste Element des Interface anzuwenden.

Bei der Analyse der Interface-Deklaration interpretiert der Compiler das erste Konstrukt in eckigen Klammern als zum GUID-Bezeichner des Interface gehörig. Die GUID muss in der herkömmlichen Delphi-Form angegeben werden:

```
[ '{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}' ]
```

Alternativ kann das benutzerdefinierte Attribut `Guid` des .NET Framework (`GuidAttribute`) verwendet werden. Bei diesem Verfahren muss das Attribut (wie jedes andere benutzerdefinierte Attribut) vor dem Interface angegeben werden.

Beide Verfahren haben dieselbe Wirkung: die GUID wird in die Metadaten des Interface-Typs ausgegeben. Im .NET Framework sind für Interfaces keine GUIDs erforderlich. Sie dienen lediglich der COM-Interoperabilität.

Anmerkung: Beim Importieren von COM-Interfaces mit dem benutzerdefinierten Attribut `ComImport` muss anstelle der Delphi-Syntax eine Deklaration von `GuidAttribute` verwendet werden.

4

Siehe auch

[Platform Invoke mit Delphi 2005 verwenden](#)

4.1.2 Liste der Compiler-Direktiven

Das folgende Thema führt alle Compiler-Direktiven von RAD Studio auf.

4.1.2.1 Felder ausrichten (Delphi)

Typ	Option
Syntax	<code>{\$A+}</code> , <code>{\$A-}</code> , <code>{\$A1}</code> , <code>{\$A2}</code> , <code>{\$A4}</code> oder <code>{\$A8}</code> <code>{\$ALIGN ON}</code> , <code>{\$ALIGN OFF}</code> , <code>{\$ALIGN 1}</code> , <code>{\$ALIGN 2}</code> , <code>{\$ALIGN 4}</code> oder <code>{\$ALIGN 8}</code>
Vorgabe	<code>{\$A8} {\$ALIGN 8}</code>
Bereich	Lokal

Anmerkungen

Die Direktive `$A` steuert die Ausrichtung von Feldern in Record-Typen und Klassenstrukturen.

Im Status `{$A1}` oder `{$A-}` werden Felder nicht ausgerichtet. Alle Records und Klassenstrukturen werden gepackt.

Im Status `{$A2}` werden die ohne den Bezeichner **packed** deklarierten Felder in Record-Typen und die Felder in Klassenstrukturen auf Wortgrenzen ausgerichtet.

Im Status `{$A4}` werden die ohne den Bezeichner **packed** deklarierten Felder in Record-Typen und die Felder in Klassenstrukturen auf Doppelwortgrenzen ausgerichtet.

Im Status `{$A8}` oder `{$A+}` werden die ohne den Bezeichner **packed** deklarierten Felder in Record-Typen und die Felder in Klassenstrukturen auf Vierfachwortgrenzen ausgerichtet.

Die Ausrichtung von Feldern in Record-Typen wird in der Delphi-Sprachreferenz beschrieben.

Variablen und typisierte Konstanten werden unabhängig von der Direktive `$A` immer für einen optimalen Zugriff ausgerichtet. Der Status `{$A8}` beschleunigt die Ausführung.

Siehe auch

[Record-Typen \(siehe Seite 902\)](#)

4.1.2.2 Anwendungstyp (Delphi)

Typ	Parameter
Syntax	{\$APPTYPE GUI} oder {\$APPTYPE CONSOLE}
Vorgabe	{\$APPTYPE GUI}
Bereich	Global

Anmerkungen

Diese Direktive wird nur bei der Delphi-Windows-Programmierung verwendet.

Die Direktive \$APPTYPE steuert, ob eine Win32-Konsolenanwendung oder eine grafische UI-Anwendung erzeugt wird.

Im Status {\$APPTYPE GUI} erzeugt der Compiler eine grafische UI-Anwendung. Dies ist der normale Status einer Delphi-Anwendung.

Im Status {\$APPTYPE CONSOLE} (entspricht der Kommandozeilenoption /CC) erzeugt der Compiler eine Konsolenanwendung. Die Standardtextdateien für Ein- und Ausgabe werden dem Konsolenfenster in einer Konsolenanwendung automatisch zugeordnet.

Die Einstellung {\$APPTYPE CONSOLE} eignet sich für das Debuggen, da Sie dabei WriteLn-Anweisungen in Ihrem Programm verwenden können, ohne explizit eine Ausgabedatei zu öffnen.

Anhand der in der System-Unit deklarierten Boolesche Variablen IsConsole können Sie feststellen, ob ein Programm als Konsolenanwendung oder als grafische UI-Anwendung ausgeführt wird.

Die Direktive \$APPTYPE ist nur in einem Programm sinnvoll. Sie sollte nicht in einer Bibliothek, Unit oder einem Package verwendet werden.

4.1.2.3 Assert-Direktiven (Delphi)

Typ	Option
Syntax	{\$C+} oder {\$C-} {\$ASSERTIONS ON} oder {\$ASSERTIONS OFF}
Vorgabe	{\$C+} {\$ASSERTIONS ON}
Bereich	Lokal

Anmerkungen

Die Direktive \$C aktiviert oder deaktiviert die Generierung von Assert-Code in einer Quelltextdatei. {Die Voreinstellung ist {\$C+}.}

Gewöhnlich werden in der Auslieferungsversion eines Produkts zur Laufzeit keine Asserts verwendet. {Deshalb steht die Compiler-Direktive {\$C-} zur Verfügung, mit der die Erzeugung von entsprechendem Code deaktiviert werden kann.}

4.1.2.4 AUTOBOX-Direktive (Delphi für .NET)

Typ	Option
Syntax	{\$AUTOBOX ON}, {\$AUTOBOX OFF}
Vorgabe	{\$AUTOBOX OFF}

Bereich	Lokal
---------	-------

Anmerkungen

Die Direktive **\$AUTOBOX** steuert, ob Wertetypen automatisch in Referenztypen umgewandelt werden.

Der folgende Code wird standardmäßig nicht kompiliert; der Compiler hält mit der Meldung an, dass `I` und `Obj` inkompatible Typen haben.

```
4
var
  I: Integer;
  Obj: TObject;
begin
  I := 5;
  Obj:=I; // Compilierfehler
end.
```

Fügen Sie **{\$AUTOBOX ON}** irgendwo vor der fehlerhaften Zeile ein. Der Fehler wird dadurch entfernt und der folgende Code wird kompiliert:

```
var
  I: Integer;
  Obj: TObject;
begin
  I := 5;
  {$AUTOBOX ON}
  Obj:=I; // I wird in ein TObject umgewandelt
end.
```

Referenztypen können mit AUTOBOX nicht automatisch in Wertetypen umgewandelt werden. Zum Konvertieren eines TObject in ein Integer ist eine Typumwandlung erforderlich:

```
var
  I: Integer;
  Obj: TObject;
begin
  I := 5;
  {$AUTOBOX ON}
  Obj:=I; // OK
  // I:=Obj; // Kann nicht automatisch umgewandelt werden; Compilierfehler
  I:=Integer(Obj); // das funktioniert
end.
```

Das Aktivieren von AUTOBOX ist praktisch, macht Delphi aber unsicherer, daher kann es gefährlich sein. Bei aktiviertem AUTOBOX könnten Fehler, die sonst während des Compilierens abgefangen werden, zur Laufzeit Probleme verursachen. Das Umwandeln von Werten in Objektreferenzen verbraucht außerdem zusätzlichen Arbeitsspeicher und vermindert die Ausführungsgeschwindigkeit. Bei **{\$AUTOBOX ON}** laufen Sie Gefahr, nicht zu bemerken, wie viel von dieser Datenumwandlung in Ihrem Quelltext ausgeführt wird. **{\$AUTOBOX OFF}** wird für eine verbesserte Typprüfung und schnellere Ausführungsgeschwindigkeit empfohlen.

Die Direktive **\$AUTOBOX** hat in Delphi für Win32 keine Auswirkungen.

4.1.2.5 Boolesche Kurzauswertung (Delphi-Compiler-Direktive)

Typ	Option
Syntax	<code>{\$B+}</code> oder <code>{\$B-}</code> <code>{\$BOOLEVAL ON}</code> oder <code>{\$BOOLEVAL OFF}</code>
Vorgabe	<code>{\$B-}</code> <code>{\$BOOLEVAL OFF}</code>
Bereich	Lokal

Anmerkungen

Die Direktive `$B` schaltet zwischen zwei unterschiedlichen Modellen der Code-Generierung für die Booleschen Operatoren **and** und **or** um.

Im Status `{$B+}` erzeugt der Compiler Code für die vollständige Auswertung eines Booleschen Ausdrucks. Das bedeutet, dass jeder Operand eines Booleschen Ausdrucks, der mit den Operatoren `and` und `or` gebildet wird, garantiert ausgewertet wird, auch wenn das Ergebnis des gesamten Ausdrucks bereits feststeht.

Im Status `{$B-}` generiert der Compiler Code für die Kurzschlussauswertung Boolescher Ausdrücke, d. h. die Auswertung wird beendet, sobald das Ergebnis des gesamten Ausdrucks feststeht (die Auswertung erfolgt immer von links nach rechts).

Siehe auch

Boolesche Operatoren (siehe Seite 877)

4

4.1.2.6 Debug-Informationen (Delphi)

Typ	Option
Syntax	<code>{\$D+}</code> oder <code>{\$D-}</code> <code>{\$DEBUGINFO ON}</code> oder <code>{\$DEBUGINFO OFF}</code>
Vorgabe	<code>{\$D+}</code> <code>{\$DEBUGINFO ON}</code>
Bereich	Global

Anmerkungen

Mit der Direktive `$D` kann die Generierung von Debug-Informationen aktiviert und deaktiviert werden. Diese Informationen beinhalten für jede Prozedur eine Tabelle mit Zeilennummern, in der Adressen des Objektcodes als Zeilennummern im Quelltext dargestellt werden.

Bei Units werden die Debug-Informationen in der Unit-Datei gemeinsam mit dem Objektcode der Unit aufgezeichnet. Durch die Debug-Informationen erhöht sich die Größe der Unit-Datei. Das Compilieren von Programmen, die diese Unit verwenden, erfordert deshalb mehr Speicher. Die Größe und die Ausführungsgeschwindigkeit des ausführbaren Programms werden aber nicht nachteilig beeinflusst.

Wenn ein Programm oder eine Unit im Status `{$D+}` compiliert wird, können Sie das betreffende Modul mit dem integrierten Debugger in Einzelschritten testen und Haltepunkte setzen.

Mit den Optionen **Mit Debug-Infos** und **Map-Datei** (auf der Registerkarte **Linker** des Dialogfelds **Projektoptionen**) können nur vollständige Zeileninformationen für Module erzeugt werden, die im Status `{$D+}` compiliert wurden.

Der Schalter `$D` wird normalerweise zusammen mit dem Schalter `$L` eingesetzt, der die Erzeugung von lokalen Symbolinformationen für das Debugging steuert.

4.1.2.7 DEFINE-Direktive (Delphi)

Typ	Bedingte Compilierung
Syntax	<code>{\$DEFINE Name}</code>

Anmerkungen

Diese Direktive definiert ein bedingtes Symbol mit dem angegebenen Namen. Das Symbol ist bekannt, bis es in einer `{$UNDEF Name}`-Direktive auftaucht oder bis das aktuelle Modul vollständig compiliert ist. Die Direktive `{$DEFINE Name}` hat keine Auswirkungen, wenn Name bereits definiert ist.

4.1.2.8 DENYPACKAGEUNIT-Direktive (Delphi)

Typ	Option
Syntax	{\$DENYPACKAGEUNIT ON} oder {\$DENYPACKAGEUNIT OFF}
Vorgabe	{\$DENYPACKAGEUNIT OFF}
Bereich	Lokal

Anmerkungen

Die Direktive {\$DENYPACKAGEUNIT ON} verhindert, dass die entsprechende Unit in ein Package eingefügt wird.

4.1.2.9 Beschreibung (Delphi)

Typ	Parameter
Syntax	{\$DESCRIPTION 'Text'}
Bereich	Global

Anmerkungen

Die Direktive \$D fügt den angegebenen Text als Beschreibung in den Header einer ausführbaren Datei, DLL- oder Package-Datei ein. Bei dem Text handelt es sich normalerweise um einen Namen oder eine Versionsnummer. Sie können aber beliebigen Text angeben. Ein Beispiel:

```
{$D 'Meine Anwendung, Version 12.5'}
```

Der String ist auf eine Länge von 256 Byte beschränkt. Der Text muss in halbe Anführungszeichen eingeschlossen werden und bleibt dem Endbenutzer normalerweise verborgen. Mithilfe von Versionsinfo-Ressourcen können Sie Ihre Anwendung mit beschreibendem Text sowie Versions- und Copyright-Informationen versehen, die auch für den Endbenutzer sichtbar sind.

Anmerkung: Der Beschreibungstext muss in Anführungszeichen gesetzt werden.

4.1.2.10 DESIGNONLY-Direktive (Delphi)

Typ	Option
Syntax	{\$DESIGNONLY ON} oder {\$DESIGNONLY OFF}
Vorgabe	{\$DESIGNONLY OFF}
Bereich	Lokal

Anmerkungen

Die Direktive {\$DESIGNONLY ON} bewirkt, dass das entsprechende Package für die Installation in der Entwicklungsumgebung compiliert wird.

Verwenden Sie die Direktive DESIGNONLY nur in .dpk-Dateien.

Siehe auch

Spezielle Compiler-Direktiven für Packages (siehe Seite 1002)

4.1.2.11 ELSE (Delphi)

Typ	Bedingte Compilierung
Syntax	{\$ELSE}

Anmerkungen

Mit dieser Direktive kann zwischen Compilieren und Ignorieren des Quelltextes, der zwischen der vorhergehenden {\$IFxxx}-Direktive und der nächsten {\$ENDIF}- oder {\$IFEND}-Direktive steht, umgeschaltet werden.

4.1.2.12 ELSEIF (Delphi)

Typ	Bedingte Compilierung
Syntax	{\$ELSEIF}

Anmerkungen

Mit **\$ELSEIF** können Sie mehrteilige bedingte Blöcke definieren, von denen höchstens ein Block ausgeführt wird. **\$ELSEIF** ist eine Kombination von **\$ELSE** und **\$IF**.

Ein Beispiel:

```
{$IFDEF foobar}
  do_foobar
  {$ELSEIF RTLVersion >= 14}
    blah
  {$ELSEIF somestring = 'Ja'}
    beep
  {$ELSE}
    last chance
  {$IFEND}
```

Von diesen vier Blöcken wird jeweils nur einer ausgeführt. Trifft keine der ersten drei Bedingungen zu, werden die Anweisungen nach der Klausel **\$ELSE** ausgeführt. **\$ELSEIF** muss durch die Klausel **\$IFEND** abgeschlossen werden. **\$ELSEIF** kann nicht nach **\$ELSE** angegeben werden. Die Bedingungen werden wie bei einer normalen if ... else-Verzweigung von oben nach unten ausgewertet. Wenn im Beispiel foobar nicht definiert ist, RTLVersion den Wert 15 hat und somestring = 'Ja' ist, wird nur der Block blah und nicht beep ausgeführt, obwohl beide Bedingungen wahr sind.

4.1.2.13 ENDIF-Direktive

Typ	Bedingte Compilierung
Syntax	{\$ENDIF}

Anmerkungen

Diese Direktive beendet die bedingte Compilierung, die mit der letzten {\$IFxxx}-Direktive begonnen wurde.

Anmerkung: Schließen Sie die Direktiven **\$IF** und **\$ELSEIF** mit **\$IFEND** anstelle von **\$ENDIF** ab.

4.1.2.14 Erweiterung für ausführbare Dateien (Delphi)

Typ	Parameter
Syntax	{\$E Erweiterung} {\$EXTENSION Erweiterung}

Mit der Direktive **\$\$E** wird die Dateinamenserweiterung für die vom Compiler generierte ausführbare Datei festgelegt. Sie wird oft zusammen mit dem nur für Ressourcen-Dateien geltenden Mechanismus für gemeinsame Objekte verwendet.

Wenn Sie beispielsweise in einem Bibliotheksmodul die Direktive **{\$E deu}** angeben, wird ein gemeinsames Objekt mit der Erweiterung .deu (also Dateiname.deu) erzeugt. Dateiname.de) erzeugt. Wenn Sie ein Bibliotheksmodul erstellen, das nur auf deutsche Formulare und Strings verweist, so können Sie mit dieser Direktive ein gemeinsames Objekt mit der Erweiterung .deu erzeugen. Der Start-Code in der Laufzeitbibliothek sucht nach einem gemeinsamen Objekt mit einer Erweiterung, die mit der landesspezifischen Systemkennung übereinstimmt. Bei Einstellungen für Deutschland wird nach .deu gesucht, und anschließend werden die Ressourcen aus der betreffenden Datei geladen.

4.1.2.15 Symbole exportieren (Delphi)

Typ	Option
Syntax	{\$ObjExportAll On} oder {\$ObjExportAll Off}
Vorgabe	{\$ObjExportAll Off}
Bereich	Global

Die Anweisung **{\$ObjExportAll On}** dient zum Exportieren aller Symbole in der Unit-Datei, in der sie enthalten ist. Dadurch kann der C++ Compiler Packages mit Objektdateien erzeugen, die von Delphi generiert wurden.

4.1.2.16 Erweiterte Syntax (Delphi)

Typ	Option
Syntax	{\$X+} oder {\$X-} {\$EXTENDEDNTAX ON} oder {\$EXTENDEDNTAX OFF}
Vorgabe	{\$X-} {\$EXTENDEDNTAX ON}
Bereich	Global

Anmerkungen

Anmerkung: Die Direktive **\$X** dient lediglich der Abwärtskompatibilität. Verwenden Sie daher **{\$X-}** nicht in Ihren Delphi-Anwendungen.

Mit der Direktive **\$X** lässt sich die erweiterte Syntax von Delphi aktivieren oder deaktivieren:

- Funktionsanweisungen Im Modus **{\$X+}** lassen sich Funktionsaufrufe als Prozeduraufufe verwenden, d.h. das Ergebnis eines Funktionsaufrufs kann ignoriert werden, anstatt an eine andere Funktion übergeben oder in einer Operation bzw. Zuweisung verwendet zu werden. Im Allgemeinen werden die von einer Funktion ausgeführten Berechnungen durch das Funktionsergebnis repräsentiert, das nicht ignoriert werden sollte. Manchmal führen Funktionen aber lediglich eine bestimmte Operation durch (z. B. einer globalen Variablen einen Wert zuweisen) und geben keinen Wert zurück, der weiterverwendet werden kann.
- Result-Variable Im Modus **{\$X+}** kann die vordefinierte Variable **Result** für den Rückgabewert der Funktion verwendet werden.

werden.

- Nullterminierte Strings Im Modus {\$X+} können Delphi-Strings nullbasierten Zeichen-Arrays (**array[0..X] of Char**) zugewiesen werden, die mit den PChar-Typen kompatibel sind.

Siehe auch

Funktionsdeklarationen ([siehe Seite 931](#))

Nullterminierte Strings ([siehe Seite 896](#))

4.1.2.17 Externe Symbole (Delphi)

Typ	Parameter
Syntax	{\$EXTERNALSYM Bezeichner}

Die Direktive EXTERNALSYM verhindert, dass das angegebene Delphi-Symbol in für C++ generierten Header-Dateien enthalten ist. Wenn eine überladene Routine angegeben wird, werden alle Versionen der Routine aus der Header-Datei ausgeschlossen.

4.1.2.18 Prüfung von Fließkomma-Exceptions (Delphi)

Typ	Option
Syntax	{\$FINITEFLOAT ON}, {\$FINITEFLOAT OFF}
Vorgabe	{\$FINITEFLOAT ON}
Bereich	Global

Anmerkungen

Die Direktive **\$FINITEFLOAT** steuert die Behandlung von Fließkommaüber- und unterläufen und ungültigen Fließkommaoperationen, wie z.B. die Division durch Null.

Im Status **{\$FINITEFLOAT ON}** (Vorgabe) werden die Ergebnisse von Fließkommaberechnungen geprüft und eine Exception bei Überlauf, Unterlauf oder einer ungültigen Operation ausgelöst. Im Status **{\$FINITEFLOAT OFF}** geben solche Fließkommaberechnungen NAN, -INF oder +INF zurück.

Für die Überprüfung der Ergebnisse von Fließkommaberechnungen und dem Auslösen von Exceptions wird zur Laufzeit zusätzliche Verarbeitungskapazität benötigt. Wenn in Ihrem Delphi-Quelltext Fließkommaoperationen verwendet werden, aber das strikte Erzwingen von Überlauf-/Unterlauf-Exceptions nicht erforderlich ist, können Sie **{\$FINITEFLOAT OFF}** setzen, damit die Ausführung etwas schneller vonstatten geht.

Anmerkung: Der meiste Quelltext in .NET wird ohne Fließkommaprüfungen ausgeführt. Delphi stellt jedoch traditionell eine strikte Fließkommamantik bereit. Wenn für Ihren Delphi-Quelltext wichtig ist, dass Exceptions bei Überlauf- und Unterlaufbedingungen ausgelöst werden, sollten Sie die Standardeinstellung **({\$FINITEFLOAT ON})** beibehalten.

Siehe auch

Interne Datenformate

4.1.2.19 Hinweise (Delphi)

Typ	Option
Syntax	{\$HINTS ON} oder {\$HINTS OFF}
Vorgabe	{\$HINTS ON}
Bereich	Lokal

Anmerkungen

Die Direktive **\$HINTS** steuert die Erzeugung von Hinweisen durch den Compiler.

Im Status **{\$HINTS ON}** gibt der Compiler Hinweise aus, wenn er Variablen, Zuweisungen und **for**- oder **while**-Schleifen findet, die nicht verwendet werden oder nie zur Ausführung kommen. Im Status **{\$HINTS OFF}** erzeugt der Compiler keine solchen Hinweise.

Durch Einfügen von Quelltext zwischen **{\$HINTS OFF}** und **{\$HINTS ON}** können Sie selektiv die Generierung von Hinweisen deaktivieren, die nicht benötigt werden. Zum Beispiel:

```
{$HINTS OFF}
procedure Test;
var
  I: Integer;
begin
end;
{$HINTS ON}
```

Aufgrund der **\$HINTS**-Direktiven erzeugt der Compiler beim Compilieren der obigen Prozedur keine Hinweise auf die nicht verwendete Variable.

4.1.2.20 HPP-Ausgabe (Delphi)

Typ	Parameter
Syntax	{\$HPPEMIT 'String'}

Die Direktive **HPPEMIT** fügt der für C++ generierten Header-Datei ein bestimmtes Symbol hinzu. Beispiel: **{\$HPPEMIT 'typedef double Weight' }**.

HPPEMIT-Direktiven werden im Benutzerabschnitt am Beginn der Header-Datei ausgegeben, und zwar in der Reihenfolge, in der sie in der Delphi-Datei stehen.

4.1.2.21 IFDEF-Direktive (Delphi)

Typ	Bedingte Compilierung
Syntax	{\$IFDEF Name}

Anmerkungen

Diese Direktive compiliert den nachfolgenden Quelltext, wenn Name definiert ist.

4.1.2.22 IF-Direktive (Delphi)

Typ	Bedingte Compilierung
Syntax	{\$IF Ausdruck}

Anmerkungen

Der nachfolgende Quelltext wird kompiliert, wenn Ausdruck True ist. Der Ausdruck muss der Delphi-Syntax entsprechen und einen Booleschen Wert zurückgeben. Er kann deklarierte Konstanten, konstante Ausdrücke sowie die Funktionen defined und declared enthalten.

Zum Beispiel:

```
{$DEFINE CLX}
  const LibVersion = 2.1;
  {$IF Defined(CLX) and (LibVersion > 2.0) }
    ...
    // Diese Anweisungen werden ausgeführt
  {$ELSE}
    ...
    // Diese Anweisungen werden nicht ausgeführt
  {$IFEND}
  {$IF Defined(CLX) }
    ...
    // Diese Anweisungen werden ausgeführt
  {$ELSEIF LibVersion > 2.0}
    ...
    // Diese Anweisungen werden nicht ausgeführt
  {$ELSEIF LibVersion = 2.0}
    ...
    // Diese Anweisungen werden nicht ausgeführt
  {$ELSE}
    ...
    // Diese Anweisungen werden nicht ausgeführt
  {$IFEND}
  {$IF Declared(Test)}
    ...
    // erfolgreich
  {$IFEND}
```

Die speziellen Funktionen defined und declared können nur in \$IF- und \$ELSEIF-Blöcken verwendet werden. defined gibt True zurück, wenn als Parameter ein definiertes bedingtes Symbol übergeben wird. declared gibt True zurück, wenn als Parameter ein gültiger deklarierter Delphi-Bezeichner übergeben wird, der im aktuellen Gültigkeitsbereich sichtbar ist.

Wenn die im bedingten Ausdruck verwendeten Bezeichner nicht vorhanden sind, wird die Bedingung als False ausgewertet:

```
{$IF NoSuchVariable > 5}
  WriteLn('Diese Zeile wird nicht kompiliert');
{$IFEND}
```

Die Direktiven \$IF und \$ELSEIF werden im Gegensatz zu anderen bedingten Anweisungen, die mit \$ENDIF beendet werden, durch \$IFEND abgeschlossen. Dadurch können die \$IF-Blöcke der früheren Compiler-Versionen (die \$IF oder \$ELSEIF nicht unterstützen) verborgen werden, indem sie in \$IFDEF-Blöcke eingeschlossen werden. So führt beispielsweise folgendes Konstrukt nicht zu einem Compiler-Fehler:

```
{$UNDEF NewEdition}
{$IFDEF NewEdition}
  {$IF LibVersion > 2.0}
    ...
  {$IFEND}
{$ENDIF}
```

\$IF unterstützt zwar die Auswertung typisierter Konstanten, der Compiler erlaubt diese aber nicht in konstanten Ausdrücken. Daher ist

```
const Test: Integer = 5;
{$IF SizeOf(Test) > 2}
  ...
{$ENDIF}
```

gültig, aber

```
const Test: Integer = 5;
{$IF Test > 2 }           // Fehler
...
```

führt deshalb zu einem Compilierungsfehler.

Wenn Ihr Quelltext zwischen den verschiedenen Versionen von Delphi oder anderen Plattformen (wie z.B. .NET) portierbar sein soll, können Sie testen, ob diese Direktive vom Compiler unterstützt wird. Schließen Sie den Quelltext dazu in folgende Direktiven ein:

```
$IFDEF conditionalexpressions
.           // Code mit IF-Direktive
.           // wird nur ausgeführt, wenn unterstützt
$ENDIF
```

Anmerkung: Um zu testen, ob Quelltext auf der .NET-Plattform compiliert wird, verwenden Sie den Bezeichner `CLR`, z.B. `{$IF NOT DEFINED(CLR)}`.

4.1.2.23 IFEND-Direktive (Delphi)

Typ	Bedingte Compilierung
Syntax	<code>{\$IFEND}</code>

Anmerkungen

`$IFEND` schließt die Direktiven `$IF` und `$ELSEIF` ab. Dadurch können `$IF/$IFEND`-Blöcke vor älteren Compilern in einem `$IFDEF/$ENDIF` verborgen werden, da diese `$IFEND` nicht als Direktive erkennen. `$IF` kann nur durch die Direktive `$IFEND` abgeschlossen werden. Die Direktiven `$IFDEF`, `$IFNDEF` und `$IFOPT` können nur mit `$ENDIF` abgeschlossen werden.

Anmerkung: Wenn Sie `$IF` in einem `$IFDEF/$ENDIF`-Block verbergen, verwenden Sie nicht `$ELSE` zusammen mit `$IF`. Die älteren Compiler-Versionen interpretieren `$ELSE` als Teil von `$IFDEF`, und dies führt zu einem Compiler-Fehler. Sie können in dieser Situation `{$ELSEIF True}` als Ersatz für `{$ELSE}` verwenden, da `$ELSEIF` nicht nach einem vorhergehenden `$IF` ausgewertet wird und die älteren Compiler `$ELSEIF` nicht interpretieren. Das Verbergen von `$IF` aus Gründen der Abwärtskompatibilität wird hauptsächlich von Fremdherstellern und Anwendungsentwicklern genutzt, die ihren Quelltext für verschiedene Versionen von Delphi und anderen Plattformen erstellen möchten.

4.1.2.24 IFNDEF-Direktive (Delphi)

Typ	Bedingte Compilierung
Syntax	<code>{\$IFNDEF Name}</code>

Anmerkungen

Diese Direktive compiliert den nachfolgenden Quelltext, wenn Name nicht definiert ist.

4.1.2.25 IFOPT-Direktive (Delphi)

Typ	Bedingte Compilierung
Syntax	{\$IFOPT Schalter}

Anmerkungen

Diese Direktive compiliert den nachfolgenden Delphi-Quelltext, wenn sich Schalter aktuell im angegebenen Status befindet. Das Argument Schalter besteht aus dem Namen einer Schalteroption und einem Pluszeichen (+) oder einem Minuszeichen (-). Zum Beispiel:

```
{$IFOPT R+}
  Writeln('Mit Bereichsprüfung compiliert');
{$ENDIF}
```

Das folgende Konstrukt compiliert beispielsweise die Writeln-Anweisung, wenn die Option \$R aktuell aktiv ist:

4.1.2.26 Image-Basisadresse

Typ	Parameter
Syntax	{\$IMAGEBASE Zahl}
Vorgabe	{\$IMAGEBASE \$00400000}
Bereich	Global

Die Direktive \$IMAGEBASE definiert die Standardladeadresse für eine Anwendung, DLL oder ein Package. Das Argument Zahl muss ein 32-Bit-Integerwert sein, der die Image-Basisadresse angibt. Zahl muss gleich oder größer sein als \$00010000; die geringwertigen 16 Bit des Arguments werden ignoriert oder sollten Nullen sein. Zahl muss ein Vielfaches von 64K sein (die entsprechende Hexadezimalzahl muss also mit vier Nullen enden); ist das nicht der Fall, runden der Compiler die Zahl auf das nächstkleinere Vielfache ab und gibt eine Meldung aus.

Wird ein Modul (Anwendung oder Bibliothek) in den Adressraum eines Prozesses geladen, versucht Windows, das Modul an dessen Standard-Image-Basisadresse abzulegen. Sollte das nicht möglich sein, weil die vorgegebene Adresse bereits belegt ist, wird das Modul an einer Adresse abgelegt, die Windows während der Laufzeit bestimmt (Adressverschiebung).

Für eine Änderung der Image-Basisadresse einer Anwendung gibt es so gut wie nie einen Grund. Dagegen sollten Sie für eine Bibliothek mit der Direktive \$IMAGEBASE eine andere als die Standardadresse als Image-Basisadresse angeben, da die Standardadresse \$00400000 wahrscheinlich immer belegt sein wird. Der empfohlene Adressbereich für DLL-Images liegt zwischen \$40000000 und \$7FFFFFFF. Adressen in diesem Bereich stehen unter Windows NT/2000 und Windows 95/98 immer für Prozesse zur Verfügung.

Sofern Windows eine DLL oder ein Package an der zugehörigen Image-Basisadresse ablegen konnte, wird die Ladezeit verringert, weil keine Anpassungen aufgrund von Adressverschiebungen erforderlich sind. Wenn zudem der vorgegebene Adressbereich in mehreren Prozessen, die die Bibliothek benutzen, verfügbar ist, können Teile des DLL-Image-Codes von den Prozessen gemeinsam benutzt werden, wodurch sich die Ladezeit verringert und weniger Arbeitsspeicher belegt wird.

Anmerkung: Die Direktive \$IMAGEBASE überschreibt jeden Wert, der mit der Kommandozeilen-Direktive -K angegeben wurde.

4.1.2.27 Implizites Erstellen (Delphi)

Typ	Option
Syntax	{\$IMPLICITBUILD ON} oder {\$IMPLICITBUILD OFF}
Vorgabe	{\$IMPLICITBUILD ON}
Bereich	Global

Anmerkungen

Die Direktive {\$IMPLICITBUILD OFF}, die nur für Packages gedacht ist, verhindert, dass die entsprechende Quelltextdatei zu einem späteren Zeitpunkt implizit neu compiliert wird. Verwenden Sie {\$IMPLICITBUILD OFF} in DPK-Dateien, wenn Sie Packages compilieren, die sich nur selten ändern, die Low-Level-Funktionen enthalten oder deren Quelltext nicht weitergegeben wird. Verwenden Sie {\$IMPLICITBUILD OFF} nicht in Unit-Quelldateien.

4.1.2.28 Importierte Daten

Typ	Option
Syntax	{\$G+} oder {\$G-} {\$IMPORTEDDATA ON} oder {\$IMPORTEDDATA OFF}
Vorgabe	{\$G+} {\$IMPORTEDDATA ON}
Bereich	Lokal

Anmerkungen

Mit der Direktive {\$G-} wird die Erzeugung von Referenzen auf importierte Daten deaktiviert. {\$G-} optimiert den Speicherzugriff, verhindert aber, dass die entsprechende Unit Variablen in anderen Packages referenziert.

4.1.2.29 Datei einbinden (Delphi)

Typ	Parameter
Syntax	{\$I filename} {\$INCLUDE filename}
Bereich	Lokal

Anmerkungen

Die Parameter-Direktive \$I weist den Compiler an, die angegebene Datei in die Compilierung aufzunehmen. Diese Datei wird direkt nach der Direktive {\$I Dateiname} in den Text eingefügt. Die vorgegebene Namenserweiterung für die Datei ist .pas. Wenn die Datei ohne Verzeichnispfad angegeben wird, sucht Delphi die Datei in dem Verzeichnis, in dem auch das Modul liegt. Außerdem werden alle Verzeichnisse durchsucht, die im Eingabefeld Suchpfad der Registerkarte Verzeichnisse/Bedingungen im Dialogfeld Projekt/Optionen (oder mit der Option -I in der [dccil](#)-Kommandozeile) festgelegt wurden.

Wenn der Dateiname ein Leerzeichen enthält, schließen Sie ihn in halbe Anführungszeichen ein: {\$I 'Meine Datei'}.

Für die Verwendung von Include-Dateien gilt eine Einschränkung: Sie können nicht in einem Anweisungsblock eingebunden werden. Dies liegt daran, dass zwischen den Schlüsselwörtern begin und end alle Anweisungen eines Anweisungsblocks aus derselben Quelldatei stammen müssen.

4.1.2.30 E/A-Prüfung (Delphi)

Typ	Option
Syntax	<code>{\$I+}</code> oder <code>{\$I-}</code> <code>{\$IOCHECKS ON}</code> oder <code>{\$IOCHECKS OFF}</code>
Vorgabe	<code>{\$I+}</code> <code>{\$IOCHECKS ON}</code>
Bereich	Lokal

Anmerkungen

Mit der Direktive `{$I}` wird die automatische Code-Generierung aktiviert bzw. deaktiviert, die nach jedem Aufruf einer E/A-Prozedur das Ergebnis überprüft. Eine ausführliche Beschreibung der E/A-Prozeduren finden Sie in der Delphi-Sprachreferenz. Wenn eine E/A-Prozedur bei aktiverter Option ein Ergebnis ungleich 0 zurückgibt, führt dies zu einer `EInOutError`-Exception (bzw. zum Programmabbruch, wenn die Exception-Behandlung nicht aktiviert ist). Ist die Option deaktiviert, muss die E/A-Operation durch einen Aufruf von `IOResult` auf Fehler geprüft werden.

Siehe auch

Standardroutinen und E/A. (siehe Seite 989)

4.1.2.31 Direktiven für Bibliotheken und gemeinsame Objekte (Delphi)

Typ	Parameter
Syntax	<code>\$\$OVERSION 'String'</code> <code>\$\$OVERSION 'String'</code> <code>\$\$OVERSION 'String'</code>
Vorgaben	<code>\$\$OPREFIX 'lib'</code> oder <code>\$\$OPREFIX 'bpl'</code> <code>\$\$LIBSUFFIX ''</code> <code>\$\$LIBVERSION ''</code>
Bereich	Global

Anmerkungen

Mit `$$OPREFIX` kann das Standardpräfix 'lib' oder 'bpl' der Ausgabedatei außer Kraft gesetzt werden. Sie können beispielsweise den Befehl

`{$OPREFIX 'dcl'}`

für ein Entwurfszeit-Package angeben oder folgende Direktive verwenden und auf das Präfix verzichten:

`{$LIBPREFIX ''}`

Mit `$$LIBSUFFIX` kann das angegebene Suffix vor der Erweiterung an den Namen der Ausgabedatei angefügt werden.

Verwenden Sie beispielsweise die Direktive

`{$LIBSUFFIX '-2.1.3'}`

in der Datei something.pas, um folgende Bibliothek zu generieren:

something-2.1.3.dll

Mit \$LIBVERSION kann nach der Erweiterung eine zweite Erweiterung angefügt werden. Verwenden Sie beispielsweise die Direktive

{\$LIBVERSION '-2.1.3'}

in der Datei something.pas, um folgende Bibliothek zu generieren:

libsomething.dll.2.1.3

4

4.1.2.32 Objektdatei linken (Delphi)

Typ	Parameter
Syntax	{\$L Dateiname} {\$LINK Dateiname}
Bereich	Lokal

Anmerkungen

Die Direktive \$L weist den Compiler an, die angegebene Datei mit dem compilierten Programm bzw. der compilierten Unit zu linken. Wenn Sie Prozeduren und Funktionen verwenden, die als external deklariert sind, können Sie mit dieser Direktive Code linken, der in anderen Sprachen geschrieben ist. Die angegebene Datei muss eine nach Intel-Norm verschiebbare Objektdatei (.OBJ file) sein. Die vorgegebene Namenserweiterung für die Datei ist .OBJ. Wenn die Datei ohne Verzeichnispfad angegeben wird, sucht Delphi die Datei in dem Verzeichnis, in dem auch das Modul liegt. Außerdem werden alle Verzeichnisse durchsucht, die im Eingabefeld Suchpfad der Registerkarte Verzeichnisse/Bedingungen im Dialogfeld Projekt/Optionen (oder mit der Option -I in der dccil-Kommandozeile) festgelegt wurden.

Wenn der Dateiname ein Leerzeichen enthält, schließen Sie ihn in halbe Anführungszeichen ein: {\$L 'Meine Datei'}.

Weitere Einzelheiten über das Linken mit Assembler finden Sie in der Online-Hilfe.

4.1.2.33 Lokale Symbolinformationen (Delphi)

Typ	Option
Syntax	{\$L+} oder {\$L-} {\$LOCALSYMBOLS ON} oder {\$LOCALSYMBOLS OFF}
Vorgabe	{\$L+} {\$LOCALSYMBOLS ON}
Bereich	Global

Anmerkungen

Mit der Schalter-Direktive \$L kann die Generierung der lokalen Symbolinformationen aktiviert und deaktiviert werden. Lokale Symbolinformationen sind die Namen und Typen aller lokalen Variablen und Konstanten eines Moduls, also die Symbole im **implementation**-Abschnitt des Moduls und in seinen Prozeduren und Funktionen.

Bei Units werden die lokalen Symbolinformationen zusammen mit dem Objektcode in der Unit-Datei gespeichert. Lokale Symbolinformationen vergrößern eine Unit-Datei. Das Compilieren von Programmen, die diese Unit verwenden, erfordert also mehr Speicher. Die Größe und die Ausführungsgeschwindigkeit der Programme werden aber nicht nachteilig beeinflusst.

Wenn ein Programm oder eine Unit im Status {\$L+} compiliert wird, können Sie die lokalen Variablen des Moduls im integrierten Debugger überprüfen und ändern. Außerdem können die Aufrufe von Prozeduren und Funktionen des Moduls über die Option

Ansicht/Aufruf-Stack überprüft werden.

Der Schalter \$L wird normalerweise zusammen mit dem Schalter \$D eingesetzt, der die Erzeugung von Tabellen mit Zeilennummern zu Testzwecken steuert. Im Status {\$D-} wird die Direktive \$L ignoriert.

4.1.2.34 MESSAGE-Direktive (Delphi)

Syntax	{\$MESSAGE HINT WARN ERROR FATAL 'Textstring' }
--------	---

Anmerkungen

Diese Direktive ermöglicht dem Quelltext, wie der Compiler Hinweise, Warnungen und Fehlermeldungen zu generieren. Sie ähnelt den Anweisungen #emit und pragma warn in C/C++.

Der Meldungstyp (HINT, WARN, ERROR oder FATAL) ist optional. Ohne diese Angabe wird HINT verwendet. Der Textstring muss angegeben und in einfache Anführungszeichen eingeschlossen werden.

Beispiele:

```
{$MESSAGE 'Boo!' } Hinweis
  {$Message Hint 'Füttere die Katzen'} Hinweis
  {$messagE Warn 'Sieht nach Regen aus.' } Warnung
  {$Message Error 'Nicht implementiert'} Fehler, die Compilierung wird fortgesetzt
  {$Message Fatal 'Bang. Tot.' } Fehler, die Compilierung wird abgebrochen
```

4.1.2.35 \$METHODINFO-Direktive (Delphi)

Typ	Option
Syntax	{\$METHODINFO ON} oder {\$METHODINFO OFF}
Vorgabe	{\$METHODINFO OFF}
Bereich	Lokal

Die Direktive \$METHODINFO hat nur Auswirkungen, wenn die Ausführungstypinformationen (RTTI) mit der Option {\$TYPEINFO ON} aktiviert wurden. Im Status {\$TYPEINFO ON} steuert die Direktive \$METHODINFO das Erzeugen von detaillierteren Methodendeskriptoren in den RTTI-Infos für Methoden in einem Interface. Obwohl durch {\$TYPEINFO ON} einige Informationen (RTTI) für published Methoden generiert werden, sind diese doch ziemlich begrenzt. Die Direktive \$METHODINFO erzeugt viel detailliertere (und viel größere) RTTI-Infos für Methoden. Diese Informationen beschreiben, wie die Parameter der Methode an den Stack und/oder an Register übergeben werden sollen.

Die direkte Verwendung der Direktive \$METHODINFO ist in einer Anwendung nur sehr selten erforderlich. Die Methodeninformationen vergrößern die ausführbare Datei erheblich und daher ist deren allgemeiner Gebrauch nicht empfehlenswert.

Anmerkung: Der Quelltext zur Win32-Webdienstunterstützung des Delphi-Compilers verwendet Methodeninformationsdeskriptoren, um in einem Netzwerkpaket erhaltene Parameter an die Zielmethode zu übergeben. {\$METHODINFO ON} wird nur für Webdienst-Interface-Typen verwendet.

Siehe auch

Laufzeit-Typinformationen (\$TYPEINFO-Direktive) (siehe Seite 1073)

[Aufrufbare Schnittstellen im Überblick](#)

4.1.2.36 Mindestgröße für Enum-Typen (Delphi)

Typ	Parameter
Syntax	{\$Z1} oder {\$Z2} oder {\$Z4} {\$MINENUMSIZE 1} oder {\$MINENUMSIZE 2} oder {\$MINENUMSIZE 4}
Vorgabe	{\$Z1} {\$MINENUMSIZE 1}
Bereich	Lokal

Die Direktive **\$Z** legt die minimale Speichergröße für Aufzählungstypen in Delphi fest.

Ein Aufzählungstyp wird als vorzeichenloses Byte gespeichert, wenn die Aufzählung aus maximal 256 Werten besteht und der Typ im Status **{\$Z1}** (Voreinstellung) deklariert wurde. Enthält der Aufzählungstyp mehr als 256 Werte oder wurde er im Status **{\$Z2}** deklariert, wird er als vorzeichenloses Wort gespeichert. Aufzählungstypen, die im Status **{\$Z4}** deklariert wurden, werden als vorzeichenloses Doppelwort gespeichert.

{\$Z2} und **{\$Z4}** sind hilfreich, wenn eine Schnittstelle zu C und C++ Bibliotheken benötigt wird, die Aufzählungstypen in der Regel als Wörter oder Doppelwörter darstellen.

Anmerkung: **Hinweis:** Zur Abwärtskompatibilität mit früheren Versionen von Delphi und CodeGear Pascal werden auch die Direktiven **{\$Z-}** und **{\$Z+}** unterstützt. Diese entsprechen den Schaltern **{\$Z1}** bzw. **{\$Z4}**.

4.1.2.37 Offene String-Parameter (Delphi)

Typ	Option
Syntax	{\$P+} oder {\$P-} {\$OPENSTRINGS ON} oder {\$OPENSTRINGS OFF}
Vorgabe	{\$P+} {\$OPENSTRINGS ON}
Bereich	Lokal

Anmerkungen

Die Direktive **\$P** ist nur für Quelltext von Bedeutung, der im Status **{\$H-}** compiliert wurde. Sie dient der Abwärtskompatibilität mit früheren Versionen von Delphi und CodeGear Pascal. **\$P** legt die Bedeutung von Variablenparametern fest, die mit dem Schlüsselwort **string** im Status **{\$H-}** deklariert wurden. Im Status **{\$P-}** sind Variablenparameter, die mit dem Schlüsselwort **string** deklariert wurden, normale Variablenparameter. Dagegen werden sie im Status **{\$P+}** als OpenString-Parameter behandelt. Der Bezeichner **OpenString** kann unabhängig von der Einstellung der Direktive **\$P** immer zur Deklaration von OpenString-Parametern verwendet werden.

4.1.2.38 Optimierung (Delphi)

Typ	Option
Syntax	{\$O+} oder {\$O-} {\$OPTIMIZATION ON} oder {\$OPTIMIZATION OFF}
Vorgabe	{\$O+} {\$OPTIMIZATION ON}
Bereich	Lokal

Die Direktive **\$O** steuert die Code-Optimierung. Im Status **{\$O+}** führt der Compiler eine Anzahl von Code-Optimierungen durch,

indem er beispielsweise Variablen in CPU-Registern platziert, doppelte Teilausdrücke eliminiert und Induktionsvariablen generiert. Im Status {\$O-} werden diese Optimierungen nicht durchgeführt.

Außer in bestimmten Testsituationen sollte die Codeoptimierung immer aktiviert sein. Die Optimierungen des Delphi-Compilers führen zu keinerlei Änderungen der Funktionsweise des Programms. Mit anderen Worten: Der Compiler führt keine "unsicheren" Optimierungen durch, die die besondere Aufmerksamkeit des Programmierers erfordern.

Anmerkung: Die Direktive \$O kann die Optimierung nur für eine ganze Prozedur oder Funktion ein- oder ausschalten. Dies ist nicht für bestimmte Zeilen innerhalb einer Routine möglich.

4.1.2.39 Überlaufprüfung (Delphi)

Typ	Option
Syntax	{\$Q+} oder {\$Q-} {\$OVERFLOWCHECKS ON} oder {\$OVERFLOWCHECKS OFF}
Vorgabe	{\$Q-} {\$OVERFLOWCHECKS OFF}
Bereich	Lokal

Anmerkungen

Die Direktive \$Q steuert die Erzeugung von Code für die Überlaufprüfung. Im Status {\$Q+} werden bestimmte arithmetische Integer-Operationen (+, -, *, Abs, Sqr, Succ, Pred, Inc und Dec) auf einen Überlauf geprüft. Dazu wird ihnen zusätzlicher Code hinzugefügt, der sicherstellt, dass das Ergebnis innerhalb des unterstützten Bereichs liegt. Das Fehlschlagen der Überlaufprüfung führt zu einer *EIntOverflow*-Exception (bzw. zum Programmabbruch, wenn die Exception-Behandlung nicht aktiviert ist).

Der Schalter \$Q wird normalerweise zusammen mit dem Schalter \$R eingesetzt, der die Erzeugung von Bereichsprüfungscode aktiviert bzw. deaktiviert. Die Aktivierung der Überlaufprüfung vergrößert und verlangsamt ein Programm. Die aktivierte Überlausprüfung verlangsamt und vergrößert Ihr Programm. Sie sollten {\$Q+} daher nur für die Fehlersuche verwenden.

4.1.2.40 Pentium-sichere FDIV-Operationen (Delphi)

Typ	Option
Syntax	{\$U+} oder {\$U-} {\$SAFEDIVIDE ON} oder {\$SAFEDIVIDE OFF}
Vorgabe	{\$U-}
Bereich	Lokal

Die Direktive \$U steuert die Erzeugung von Fließkommamacode, der vor der fehlerhaften FDIV-Anweisung schützt, die in bestimmten (frühen) Pentium-Prozessoren enthalten ist. Windows 95, Windows NT 3.51 und spätere Versionen enthalten Code, der den Pentium-FDIV-Fehler korrigiert und das System schützt.

Im Status {\$U+} werden alle Fließkommadivisionen von einer Laufzeit-Bibliotheksroutine ausgeführt. Beim ersten Aufruf einer Fließkommadivision prüft die Routine, ob die FDIV-Anweisung des Prozessors korrekt arbeitet, und aktualisiert die in der Unit System deklarierte Variable *TestFDIV* entsprechend. Deren Wert bestimmt bei den folgenden Fließkommadivisionen, wie vorzugehen ist.

Wert Bedeutung

-1 Die FDIV-Anweisung wurde getestet und als fehlerhaft erkannt.

0 Die FDIV-Anweisung wurde noch nicht geprüft.

1 Die FDIV-Anweisung wurde getestet und als korrekt erkannt.

Mit Pentium-Prozessoren, die keine fehlerhafte FDIV-Anweisung enthalten, bewirkt {\$U+} nur eine geringfügigen Leistungsminderung. Mit fehlerhaften Pentium-Prozessoren können Fließkommadivisionen im Status {\$U+} bis zu dreimal länger dauern, liefern aber stets korrekte Ergebnisse.

Im Status {\$U+} werden Fließkommadivisionen durch In-Line-FDIV-Anweisungen ausgeführt. Sie garantieren optimale Ausführungsgeschwindigkeiten und Codegrößen, können auf fehlerhaften Pentium-Prozessoren jedoch falsche Ergebnisse liefern. Sie sollten daher den Status {\$U+} nur verwenden, wenn Sie sicher sind, dass der Code auf einem fehlerfreien Pentium-Prozessor läuft.

4

4.1.2.41 NODEFINE

Typ	Parameter
Syntax	{\$NODEFINE Bezeichner}

Die Direktive NODEFINE verhindert, dass das angegebene Symbol in die für C++ generierte Header-Datei aufgenommen wird, während bestimmte Informationen in die Objektdatei ausgegeben werden können. Wenn Sie NODEFINE verwenden, liegt es in Ihrer eigenen Verantwortung, mit HPPEMIT alle benötigten Typen zu definieren. Ein Beispiel:

```
type
  Temperature = type double;
  {$NODEFINE Temperature}
  {$HPPEMIT 'typedef double Temperature'}
```

4.1.2.42 NOINCLUDE (Delphi)

Typ	Parameter
Syntax	{\$NOINCLUDE Dateiname}

Die Direktive NOINCLUDE verhindert, dass das angegebene Symbol in die für C++ generierte Header-Datei aufgenommen wird. {\$NOINCLUDE Unit1} entfernt beispielsweise #include Unit1.

4.1.2.43 Bereichsüberprüfung

Typ	Option
Syntax	{\$R+} oder {\$R-} {\$RANGECHECKS ON} oder {\$RANGECHECKS OFF}
Vorgabe	{\$R-} {\$RANGECHECKS OFF}
Bereich	Lokal

Anmerkungen

Mit der Direktive \$R kann die Generierung von Bereichsprüfungscode aktiviert und deaktiviert werden. Im Status {\$R+} werden alle Ausdrücke, die Arrays und Strings indizieren, dahingehend überprüft, ob sie sich innerhalb der festgelegten Grenzen befinden. Der gleichen Prüfung werden alle Zuweisungen an skalare Variablen und Teilbereichsvariablen unterzogen. Das Fehlschlagen der Bereichsprüfung führt zu einer *ERangeError*-Exception (bzw. zum Programmabbruch, wenn die Exception-Behandlung nicht aktiviert ist).

Die Aktivierung der Bereichsprüfung vergrößert und verlangsamt Ihr Programm.

4.1.2.44 Real48-Kompatibilität (Delphi)

Typ	Option
Syntax	{\$REALCOMPATIBILITY ON} oder {\$REALCOMPATIBILITY OFF}
Vorgabe	{\$REALCOMPATIBILITY OFF}
Bereich	Lokal

Anmerkungen

Mit der Standardeinstellung {\$REALCOMPATIBILITY OFF} entspricht der generische Typ Real dem Typ Double.

Wenn {\$REALCOMPATIBILITY ON} eingestellt ist, entspricht der Typ Real dem Typ Real48.

Der Schalter **REALCOMPATIBILITY** ermöglicht Abwärtskompatibilität zu altem, übernommenem Quelltext, in dem der Typ Real zur Darstellung des 6 Byte langen Typs Real diente, der jetzt Real48 heißt. In neuem Quelltext sollten Sie zur Angabe eines 6 Byte langen Typs Real den Typ Real48 verwenden. Beachten Sie jedoch, dass der Typ Real48 für Delphi für .NET ungeeignet ist.

Für die meisten Zwecke ist der Real-Typ Double am besten geeignet.

4.1.2.45 Abschnitte (Delphi und C#)

Typ	Parameter
Syntax	{\$REGION '<Abschnittsbeschreibung>'} und {\$ENDREGION} [Delphi] oder #region <Abschnittsbeschreibung> und #endregion [C#]

Die Direktiven **region** und **endregion** steuern die Anzeige von ausblendbaren Abschnitten im Quelltexteditor. Diese Direktiven werden vom Compiler ignoriert.

Zum Kennzeichnen von Quelltext als Abschnitt schließen Sie ihn in die Direktiven **region** und **endregion** ein. Sie können einen Hinweis hinzufügen, der angezeigt wird, wenn der Quelltext ausgeblendet ist.

Der folgende Code demonstriert die Verwendung eines Abschnitts in Delphi:

```
{$region 'Optionaler Text, der angezeigt wird, wenn der Codeblock ausgeblendet ist'}
// Code, der ausgeblendet werden kann
{$endregion}
```

Der folgende Code demonstriert die Verwendung eines Abschnitts in C#:

```
#region Optionaler Text, der angezeigt wird, wenn der Codeblock ausgeblendet ist
// Code, der ausgeblendet werden kann
#endregion
```

4.1.2.46 Ressourcen-Datei (Delphi)

Typ	Parameter
Syntax	{\$R Dateiname} {\$RESOURCE Dateiname} {\$R *.xxx} {\$R Dateiname.res Dateiname.rc}
Bereich	Lokal

4

Anmerkungen

Die Direktive {\$R} legt den Namen einer Ressourcen-Datei fest, die in eine Anwendung oder eine Bibliothek eingebunden werden soll. Dabei muss es sich um eine Windows-Ressourcen-Datei handeln. Wenn der Dateiname ein Leerzeichen enthält, schließen Sie ihn in halbe Anführungszeichen ein: {\$R 'Meine Datei'}.

Das Symbol * hat in {\$R}-Direktiven eine spezielle Bedeutung: Es steht für den Namensstamm (ohne Erweiterung) der Quelltextdatei, in der die Direktive enthalten ist. In der Regel weist eine Ressourcen-Datei einer Anwendung (.res) denselben Namen auf wie die zugehörige Projektdatei (.dpr). In diesem Fall bewirkt die Verwendung von {\$R *.res} in der Projektdatei, dass die zugehörige Ressourcen-Datei mit der Anwendung verknüpft wird. Analog weist eine Formulardatei (.dfm oder .nfm) in der Regel denselben Namen auf wie ihre Unit-Datei (.pas). Durch Verwendung von {\$R *.nfm} in der .pas-Datei wird die entsprechende Formulardatei mit der Anwendung verknüpft.

{\$R Dateiname.res Dateiname.rc} bewirkt, dass die .rc-Datei in der Projektverwaltung enthalten ist (hierzu müssen die beiden 'Dateinamen' übereinstimmen). Wenn der Benutzer die .rc-Datei von der Projektverwaltung aus aufruft, wird der Editor für String-Tabellen aufgerufen.

Wenn Sie die Direktive {\$R Dateiname} in einer Unit verwenden, wird der angegebene Dateiname einfach in der resultierenden Unit-Datei gespeichert. Es wird zu diesem Zeitpunkt nicht geprüft, ob der Dateiname korrekt ist und eine vorhandene Datei bezeichnet.

Beim Linken einer Anwendung oder einer Bibliothek (nach dem Compilieren des Programms bzw. der Bibliotheksquelle) erfolgt die Verarbeitung aller Ressourcen-Dateien, die in den benutzten Units und im Programm bzw. in der Bibliothek angegeben sind. Dabei werden die Ressourcen aller Ressourcen-Dateien in die ausführbare Datei kopiert. Während der Ressourcenverarbeitung sucht der Linker nach den .res-Dateien. Diese Suche erfolgt in dem Verzeichnis, in dem auch das Modul mit der Direktive {\$R} liegt. Außerdem werden alle Verzeichnisse durchsucht, die im Eingabefeld **Suchpfad** der Registerkarte **Verzeichnisse/Bedingungen** im Dialogfeld **Projekt/Optionen** (bzw. mit der Option -R in der [dccil](#)-Kommandozeile) festgelegt wurden.

4.1.2.47 RUNONLY-Direktive (Delphi)

Typ	Option
Syntax	{\$RUNONLY ON} oder {\$RUNONLY OFF}
Vorgabe	{\$RUNONLY OFF}
Bereich	Lokal

Anmerkungen

Die Direktive {\$RUNONLY ON} bewirkt, dass das entsprechende Package nur für die Verwendung zur Laufzeit compiliert wird. Diese Packages können nicht als Entwurfszeit-Packages in der IDE installiert werden.

Verwenden Sie **\$RUNONLY** nur in Package-Dateien.

Siehe auch

Packages compilieren (↗ siehe Seite 1002)

4.1.2.48 Laufzeit-Typinformationen (Delphi)

Typ	Option
Syntax	{\$M+} oder {\$M-} {\$TYPEINFO ON} oder {\$TYPEINFO OFF}
Vorgabe	{\$M-} {\$TYPEINFO OFF}
Bereich	Lokal

Die Compiler-Direktive **\$M** steuert die Erzeugung der Laufzeit-Typinformationen (RTTI). Wenn eine Klasse im Status **{\$M+}** deklariert wurde oder von einer Klasse abgeleitet ist, die im Status **{\$M+}** deklariert wurde, erzeugt der Compiler Laufzeit-Typinformationen für Felder, Methoden und Eigenschaften, die in einem published-Abschnitt deklariert sind. Ist eine Klasse im Status **{\$M+}** deklariert und nicht von einer Klasse abgeleitet, die im Status **{\$M}** deklariert wurde, sind published-Abschnitte in dieser Klasse nicht erlaubt. Wenn eine Klasse als forward deklariert ist, muss in der ersten Klassendeklaration **\$M** angegeben werden.

Wenn eine Schnittstelle mit dem Schalter **\$M** deklariert wird, erzeugt der Compiler für alle Eigenschaften und Methoden Laufzeit-Typinformationen. Daher werden bei Schnittstellen alle Elemente so behandelt, als ob sie als published deklariert sind.

Anmerkung: Die in der Unit Classes der Komponentenbibliothek definierte Klasse **TPersistent** ist im Status **{\$M+}** deklariert. Aus diesem Grund werden für alle published-Abschnitte einer von **TPersistent** abgeleiteten Klasse Laufzeit-Typinformationen generiert. Die Komponentenbibliothek verwendet diese Informationen beim Speichern oder Laden von Formulardateien für den Zugriff auf die Werte der Komponenteneigenschaften. In der IDE wird anhand der Laufzeit-Typinformationen die Liste der im Objektinspektor angezeigten Eigenschaften festgelegt.

Anmerkung: Die in der Unit System definierte Schnittstelle **IInvokable** ist im Status **{\$M+}** deklariert. Daher werden für alle von **IInvokable** abgeleiteten Schnittstellen Laufzeit-Typinformationen generiert. Mithilfe der Routinen in der Unit **IntlInfo** kann auf die Typinformationen zugegriffen werden.

Die direkte Verwendung der Direktive **{\$M}** ist in einer Anwendung nur sehr selten erforderlich.

Siehe auch

Aufrufbare Schnittstellen im Überblick

\$METHODINFO-Direktive (↗ siehe Seite 1067)

4.1.2.49 Typprüfung bei Zeigern (Delphi)

Typ	Option
Syntax	{\$T+} oder {\$T-} {\$TYPEDADDRESS ON} oder {\$TYPEDADDRESS OFF}
Vorgabe	{\$T-} {\$TYPEDADDRESS OFF}
Bereich	Global

Anmerkungen

Die Direktive **\$T** legt fest, welche Zeigertypen vom Operator **@** generiert werden, und steuert deren Kompatibilität.

Im Status {\$T-} ist das Ergebnis des Operators @ immer ein Zeiger ohne Typ (Pointer), der mit allen übrigen Zeigertypen kompatibel ist. Wenn @ im Status {\$T+} für einen Variablenverweis verwendet wird, ist das Ergebnis ein typisierter Zeiger, der nur mit Pointer und den übrigen Zeigern auf den Variablenotyp kompatibel ist.

Im Status {\$T-} sind festgelegte Zeigertypen (also nicht vom allgemeinen Typ Pointer) nicht kompatibel (auch wenn sie Zeiger auf denselben Typ sind). Im Status {\$T+} sind Zeiger, die auf denselben Typ zeigen, kompatibel.

4.1.2.50 UNDEF-Direktive (Delphi)

Typ	Bedingte Compilierung
Syntax	{\$UNDEF Name}

Anmerkungen

Diese Direktive hebt die Definition eines bedingten Symbols auf. Das Symbol wird für den Rest der Compilierung der aktuellen Quelldatei oder bis zu seiner erneuten **\$DEFINE**-Direktive ignoriert. Die Direktive **\$UNDEF** hat keine Auswirkungen, wenn die Definition von Name bereits aufgehoben ist.

Bedingte Symbole, die in der Kommandozeile oder im Dialogfeld Projektoptionen definiert wurden, werden beim Beginn der Compilierung einer jeden Unit-Quelldatei neu eingeführt. Dagegen werden die in einer Unit-Quelldatei definierten bedingten Symbole verworfen, wenn der Compiler zur nächsten Unit übergeht.

4.1.2.51 Unsicherer Code (Delphi für .NET)

Typ	Option
Syntax	{\$UNSAFECODE ON} oder {\$UNSAFECODE OFF}
Vorgabe	{\$UNSAFECODE OFF}
Bereich	Lokal

Die Direktive **\$UNSAFECODE** steuert, ob das Schlüsselwort **unsafe** vom Compiler akzeptiert wird. Bei **{\$UNSAFECODE ON}** können Sie Prozeduren und Funktionen mit dem Schlüsselwort **unsafe** kennzeichnen. Zum Beispiel:

```
procedure unsafeProc; unsafe;
begin
end;
```

Anmerkung: Unsicherer Code durchläuft weder [PEVerify](#) noch irgendeine Assemblierung oder ein Delphi-Modul, das eine **unsafe** Prozedur oder Funktion aufruft.

4.1.2.52 Prüfung von Var-String (Delphi)

Typ	Option
Syntax	{\$V+} oder {\$V-} {\$VARSTRINGCHECKS ON} oder {\$VARSTRINGCHECKS OFF}
Vorgabe	{\$V+} {\$VARSTRINGCHECKS ON}
Bereich	Lokal

Anmerkungen

Die Direktive **\$V** ist nur für Delphi-Quelltext von Bedeutung, in dem kurze Strings verwendet werden. Sie dient der Abwärtskompatibilität mit früheren Versionen von Delphi und CodeGear Pascal.

Die Direktive **\$V** steuert die Typprüfung für kurze Strings, die als var-Parameter übergeben werden. Mit **{\$V+}** wird eine strenge Typprüfung durchgeführt: Formale und tatsächliche Parameter müssen identische String- Typen aufweisen. Mit **{\$V-}** kann dagegen jede Variable des Typs **ShortString** als tatsächlicher Parameter verwendet werden, auch wenn die deklarierte Maximallänge nicht mit der des formalen Parameters identisch ist.

4.1.2.53 Warnmeldungen (Delphi)

Typ	Option
Syntax	{\$WARN Bezeichner ON} oder {\$WARN Bezeichner OFF}
Vorgabe	{\$WARN ON}
Bereich	Lokal

Anmerkungen

Mit **\$WARN** können Sie steuern, welche Gruppen von Warnmeldungen angezeigt werden. Diese Warnungen beziehen sich auf Symbole oder Units mit den Hinweisdirektiven **platform**, **deprecated** und **library**.

Der Bezeichner in **\$WARN** ist optional und kann einen der folgenden Werte annehmen:

SYMBOL_PLATFORM: Alle Warnungen bezüglich der Direktive **platform** werden für Symbole der aktuellen Unit aktiviert oder deaktiviert.

SYMBOL_LIBRARY: Alle Warnungen bezüglich der Direktive **library** werden für Symbole der aktuellen Unit aktiviert oder deaktiviert.

SYMBOL_DEPRECATED: Alle Warnungen bezüglich der Direktive **deprecated** werden für Symbole der aktuellen Unit aktiviert oder deaktiviert.

UNIT_DEPRECATED: Alle Warnungen bezüglich der Direktive **deprecated** werden für eine Unit-Deklaration aktiviert oder deaktiviert.

UNIT_LIBRARY: Alle Warnungen bezüglich der Direktive **library** werden für Units aktiviert oder deaktiviert, in denen die **library**-Direktive angegeben ist.

UNIT_PLATFORM: Alle Warnungen bezüglich der Direktive **platform** werden für Units aktiviert oder deaktiviert, in denen die **platform**-Direktive angegeben ist.

Mit **\$WARN** können nur die angegebenen Warnungen aktiviert oder deaktiviert werden.

Die mit **\$WARN** inline aktivierten Warnungen betreffen nur die Unit, in der die Direktive enthalten ist. Die Warnungen werden auch erst ab der Quelltextzeile ausgegeben, die **\$WARN** enthält.

Die Direktive **\$WARNINGS** steuert die Generierung von Compiler-Warnungen.

Siehe auch

Deklarationen (siehe Seite 862)

4.1.2.54 Warnungen (Delphi)

Typ	Option
Syntax	{\$WARNINGS ON} oder {\$WARNINGS OFF}
Vorgabe	{\$WARNINGS ON}
Bereich	Lokal

Anmerkungen

Die Direktive **\$WARNINGS** steuert die Generierung von Compiler-Warnungen. Mit **\$WARN** können Sie steuern, welche Gruppen von Warnmeldungen angezeigt werden.

Im Status **{\$WARNINGS ON}** gibt der Compiler Warnungen aus, wenn er auf nicht initialisierte Variablen, fehlende Funktionsergebnisse, die Konstruktion abstrakter Objekte usw. trifft. Im Status **{\$WARNINGS OFF}** werden keine Warnmeldungen generiert.

Durch Einfügen von Quelltext zwischen **{\$WARNINGS OFF}** und **{\$WARNINGS ON}** können Sie die Generierung von überflüssigen Warnmeldungen deaktivieren.

Anmerkung: Die Direktive **\$WARNINGS** kann nur für ganze Prozeduren oder Funktionen verwendet werden. Die Angabe von **\$WARNINGS** ist nicht bei Anweisungsblöcken in einer Prozedur oder Funktion möglich.

4.1.2.55 Schwaches Packen

Typ	Option
Syntax	{\$WEAKPACKAGEUNIT ON} oder {\$WEAKPACKAGEUNIT OFF}
Vorgabe	{\$WEAKPACKAGEUNIT OFF}
Bereich	Lokal

Anmerkungen

Mit der Direktive **\$WEAKPACKAGEUNIT** können Sie steuern, wie eine .dcu-Datei in den .dcp- und .bpl-Dateien von Delphi-Packages auf der Win32-Plattform bzw. wie eine .dcu1-Datei in den .dcu1l- und .dll-Dateien von Packages auf der .NET-Plattform gespeichert wird. Die Direktive **{\$WEAKPACKAGEUNIT ON}** in einer Unit-Datei veranlasst den Compiler, diese Unit wenn möglich nicht in .bpl- oder .dll-Dateien einzubinden. Er erzeugt stattdessen eine nicht an das Package gebundene lokale Kopie der Unit, wenn diese von einer anderen Anwendung oder einem anderen Package benötigt wird. Stattdessen wird eine nicht gepackte lokale Kopie der Unit erzeugt, wenn diese von einer anderen Anwendung bzw. einem anderen Package benötigt wird.

Nehmen wir an, ein Package namens PACK enthält nur eine Unit mit dem Namen UNIT1. UNIT1 nutzt keine weiteren Units, ruft aber RARE.DLL auf. Wenn nun vor dem Compilieren die Direktive **{\$WEAKPACKAGEUNIT ON}** in UNIT1.pas eingefügt wird, führt dies dazu, dass UNIT1 nicht in PACK.BPL (oder PACK.DLL unter .NET) aufgenommen wird. Kopien von RARE.DLL müssen nicht mit PACK verteilt werden. Dagegen ist UNIT1 weiterhin in PACK.dcp (oder PACK.dcu1l unter .NET) enthalten. Wenn andere Packages oder Anwendungen, die PACK benutzen, UNIT1 referenzieren, wird sie aus PACK.dcp (oder PACK.dcu1l unter .NET) kopiert und direkt in das Projekt compiliert.

Nehmen wir nun an, dass eine weitere Unit mit dem Namen UNIT2 zu PACK hinzugefügt wird. UNIT2 greift auf UNIT1 zu. In diesem Fall wird UNIT1 selbst dann in PACK.BPL (oder PACK.DLL unter .NET) eingebunden, wenn UNIT1.pas beim Compilieren von PACK die Direktive

{\$WEAKPACKAGEUNIT ON} enthält. Alle anderen Packages oder Anwendungen, die sich auf UNIT1 beziehen, verwenden dagegen die nicht in ein Package eingebundene Kopie aus der Datei PACK.dcp (oder PACK.dcpil unter .NET).

Anmerkung: Unit-Dateien, welche die Direktive **{\$WEAKPACKAGEUNIT ON}** verwenden, dürfen keine globalen Variablen und Abschnitte zur Initialisierung und Finalisierung enthalten.

Die Direktive **\$WEAKPACKAGEUNIT** sollte nur von erfahrenen Entwicklern verwendet werden, die ihre Packages an andere Programmierer weitergeben wollen. Sie können mit dieser Direktive vermeiden, dass weniger gebräuchliche DLLs verteilt werden. Außerdem werden Konflikte zwischen Packages eliminiert, die von derselben externen Bibliothek abhängig sind.

Beispielsweise referenziert die PenWin-Unit von Delphi die Datei PENWIN.DLL. Die meisten Projekte benutzen PenWin nicht, und auf den meisten Computern ist PENWIN.DLL nicht installiert. Aus diesem Grund enthält VCL60 die PenWin-Unit nur schwach gepackt (VCL60 kapselt viele gebräuchliche Delphi-Komponenten). Beim Compilieren eines Projekts, das PenWin sowie das VCL60-Packages nutzt, wird PenWin aus VCL60.DCP kopiert und direkt in das Projekt eingebunden; die resultierende ausführbare Datei wird statisch zu PENWIN.DLL gelinkt.

Wäre PenWin nicht schwach gepackt, würden sich zwei grundsätzliche Probleme ergeben. Erstens würde VCL60 selbst statisch zu PENWIN.DLL gelinkt und könnte somit nur geladen werden, wenn PENWIN.DLL installiert ist. Zweitens würde bei dem Versuch, PenWin in ein Package einzubinden, ein Compiler-Fehler auftreten, weil die PenWin-Unit sowohl in VCL60 als auch im neuen Package enthalten wäre. Somit könnte PenWin ohne schwache Packung nicht in Standarddistributionen von VCL60 enthalten sein.

4.1.2.56 Stack-Frames (Delphi)

Typ	Option
Syntax	{\$W+} oder {\$W-} {\$STACKFRAMES ON} oder {\$STACKFRAMES OFF}
Vorgabe	{\$W-} {\$STACKFRAMES OFF}
Bereich	Lokal

Anmerkungen

Die Direktive **\$W** steuert die Erzeugung von Stack-Frames für Prozeduren und Funktionen. Im Status **{\$W+}** werden Stack-Frames für Prozeduren und Funktionen auch dann erzeugt, wenn sie nicht benötigt werden. Im Status **{\$W-}** werden Stack-Frames nur generiert, wenn die Verwendung lokaler Variablen durch die Routine dies erforderlich macht.

Normalerweise wird der Status **{\$W+}** nur für bestimmte Debugger-Tools benötigt, die eine Generierung von Stack-Frames für alle Prozeduren und Funktionen verlangen.

4.1.2.57 Schreibbare typisierte Konstanten (Delphi)

Typ	Option
Syntax	{\$J+} oder {\$J-} {\$WRITEABLECONST ON} oder {\$WRITEABLECONST OFF}
Vorgabe	{\$J-} {\$WRITEABLECONST OFF}
Bereich	Lokal

Die Direktive **\$J** legt fest, ob typisierte Konstanten geändert werden können. Im Status **{\$J+}** ist eine Änderung möglich. Typisierte Konstanten sind in diesem Fall mit initialisierten Variablen vergleichbar. Im Status **{\$J-}** sind typisierte Konstanten tatsächlich konstant. Jeder Versuch, sie zu ändern, führt zu einer Fehlermeldung durch den Compiler.

Als *schreibbar* werden typisierte Konstanten bezeichnet, die zur Laufzeit als Variablen verwendet und somit geändert werden können. Zum Beispiel:

```
const
  foo: Integer = 12;
begin
  foo := 14;
end.
```

Im Status \$WRITEABLECONST OFF führt dieser Quelltext bei der Zuweisung an die Variable foo im begin..end-Block zu einem Compiler-Fehler. Sie können das Problem beheben, indem Sie foo nicht als const, sondern als var deklarieren.

In den früheren Versionen von Delphi und CodeGear Pascal konnten typisierte Konstanten immer geändert werden (wie im Status {\$J+}). Deshalb muss älterer Quelltext, der änderbare typisierte Konstanten enthält, im Status {\$J+} compiliert werden.

4.1.2.58 PE (Portable Executable) Header-Flags (Delphi)

Typ	Flag
Syntax	{\$SetPEFlags <Integer-Ausdruck>} {\$SetPEOptFlags <Integer-Ausdruck>}
Bereich	Lokal

Microsoft verwendet PE (Portable Executable) Header-Flags, um einer Anwendung zu ermöglichen, die Kompatibilität mit BS-Services anzuzeigen oder erweiterte BS-Services anzufordern. Diese Direktiven bieten leistungsstarke Optionen für die Feinabstimmung Ihrer Anwendungen auf High-End-NT-Systemen.

Warnung: Es gibt keine Fehlerüberprüfung oder Maskierung von Bit-Werten, die von diesen Direktiven festgelegt werden. Wenn Sie eine falsche Bit-Kombination setzen, können Sie Ihre ausführbare Datei beschädigen.

Diese Direktiven ermöglichen das Setzen von Flag-Bits im Characteristics-Feld des PE-Datei-Headers bzw. im DLLCharacteristics-Feld des optionalen PE-Datei-Headers. Die meisten mit \$SetPEFlags gesetzten Characteristics-Flags sind für Objektdateien und Bibliotheken spezifisch. Mit \$SetPEOptFlags gesetzte DLLCharacteristics sind Flags, die beschreiben, wann ein DLL-Einsprungspunkt aufgerufen werden muss..

Der <Integer-Ausdruck> in diesen Direktiven kann Delphi-Konstantenbezeichner enthalten, wie z.B. die in Windows.pas definierten IMAGE_FILE_xxxx-Konstanten. Mehrere Konstanten sollten mit OR aneinander gereiht werden.

Sie können diese Direktiven mehrmals in den Quelltext aufnehmen. Die von mehreren Direktiven festgelegten Flag-Werte sind streng kumulativ. Wenn das erste Vorkommen der Direktive \$03 und das zweite \$10 setzt, beträgt der beim Linken in die ausführbare Datei geschriebene Wert \$13 (plus der Bits, die der Linker normalerweise in den PE-Flag-Feldern setzt).

Diese Direktiven wirken sich nur auf die Ausgabedatei aus, wenn sie vor dem Linken in den Quellcode aufgenommen wurden. Das bedeutet, dass Sie diese Direktiven in eine .dpr- oder .dpk-Datei einfügen sollten und nicht in eine reguläre Unit. Wie bei der EXE-Beschreibungs-Direktive ist es kein Fehler diese Direktiven in einen Unit-Quelltext zu platzieren, aber dann haben sie nur Auswirkungen auf die Ausgabedatei (exe oder dll), wenn der Unit-Quellcode neu compiliert wird, wenn die Ausgabedatei gelinkt wird.

4.1.2.59 Reservierter Adressraum für Ressourcen (Delphi)

Typ	OS	Parameter
Syntax	Linux	{\$M ReservierteBytes} {\$RESOURCERESERVE ReservierteBytes}
Vorgabe		{\$M 1048576}

Bereich	Global
---------	--------

Anmerkungen

Diese Direktive wird nur bei der Linux-Programmierung verwendet. Informationen über die Direktiven \$M (\$MINSTACKSIZE und \$MAXSTACKSIZE) finden Sie bei der Erläuterung der Speicherreservierung.

Mit **\$M** können Sie den für Ressourcen zusätzlich reservierten Adressraum vergrößern oder verkleinern.

Der Compiler reserviert standardmäßig 1 MB Adressraum zusätzlich zu dem von der Anwendung beim Linken verwendeten Speicher für Ressourcen. Dieser zusätzliche Speicher ermöglicht es, in lokalisierten Programmen größere Ressourcen-Dateien als die Originaldateien zu verwenden. Wenn ausreichend reservierter Adressraum zur Verfügung steht, muss die lokalisierte Anwendungsversion nicht neu gelinkt werden.

Siehe auch

[Speicherreservierung](#)

4.1.3 Delphi Compiler-Fehler

Die folgenden Themen beschreiben die verschiedenen Arten von Compiler-Fehlern und -Warnungen sowie die Lösungen zu Problemen, auf die Sie eventuell bei der Verwendung dieses Produkts stoßen.

4.1.3.1 Laufzeitfehler

Bei bestimmten Laufzeitfehlern werden Delphi-Programme nach der Anzeige einer Fehlermeldung beendet.

Laufzeitfehler identifizieren

Laufzeitfehler haben folgende Form:

Laufzeitfehler nnn bei xxxxxxxx

Hierbei steht "nnn" für die Nummer des Laufzeitfehlers und "xxxxxxx" für die Fehleradresse.

Wenn eine Anwendung die Klasse SysUtils verwendet, werden die meisten Laufzeitfehler in Exceptions umgesetzt, die programmseitig gelöst werden können, ohne dass die Anwendung beendet werden muss.

Arten von Laufzeitfehlern

In Delphi gibt es folgende Arten von Laufzeitfehlern:

- E/A-Fehler (100 bis 149)
- Schwerwiegende Fehler (200 bis 255)
- Betriebssystemfehler

Siehe auch

[Exceptions \(siehe Seite 977\)](#)

[Interne Fehler beheben \(siehe Seite 1080\)](#)

[Schwerwiegende Fehler \(siehe Seite 1081\)](#)

[E/A-Fehler \(siehe Seite 1083\)](#)

[Betriebssystemfehler \(siehe Seite 1083\)](#)

4.1.3.2 Interne Fehler beheben

Eine Fehlermeldung der Form `Interner Fehler: X1234` bedeutet, dass der Compiler auf eine Bedingung gestoßen ist, die keinen Syntaxfehler darstellt und die nicht erfolgreich bearbeitet werden kann.

Tip: Die Nummernangabe bei internen Fehlern bezeichnet die Datei und die Zeilennummer im Compiler, in welcher der Fehler aufgetreten ist. Diese Informationen erleichtern dem technischen Support die Eingrenzung des Problems. Fügen Sie diese Informationen daher immer Ihrer Fehlerbeschreibung bei.

So beheben Sie einen internen Fehler:

1. Wenn der Fehler unmittelbar nach einer Quelltextänderung im Editor auftritt, kehren Sie zu der betreffenden Stelle im Quelltext zurück, und notieren Sie, was geändert wurde.
2. Machen Sie die Änderungen rückgängig, oder kommentieren Sie sie aus. Wenn die Anwendung nach dieser Maßnahme erfolgreich compiliert werden kann, ist anzunehmen, dass das betreffende Programmkonstrukt den Fehler verursacht. Falls dies so ist, überprüfen Sie zuerst den Quelltext, und führen Sie dann folgende Schritte durch.

Wenn das Problem bestehen bleibt, gehen Sie folgendermaßen vor:

1. Löschen Sie alle `.dcu1`-Dateien, die mit dem Projekt verknüpft sind.
2. Schließen das ganze Projekt mit **Datei**▶**Alle schließen**.
3. Öffnen Sie das Projekt erneut. Dadurch wird der Inhalt des Unit-Puffers der IDE gelöscht. Sie können auch die IDE schließen und neu starten.
4. Eine weitere Möglichkeit besteht darin, die Anwendung mit dem Befehl **Projekt**▶**<Projektname> erzeugen** probehalber neu zu compilieren, damit der Compiler alle `.dcu1`-Dateien neu erstellt.
5. Wenn der Fehler bestehen bleibt, verlassen Sie die IDE und compilieren die Anwendung von einer MS-DOS-Eingabeaufforderung aus mit der Befehlszeilenversion des Compilers (`dccl.exe`) erneut. Bei dieser Maßnahme wird der Unit-Zwischenspeicher der IDE gelöscht, wodurch das Problem möglicherweise behoben ist.

Überprüfen Sie den Quelltext an der letzten Änderungsstelle:

1. Wenn das Problem immer noch besteht, kehren Sie in der Datei zu der Stelle zurück, an der Sie die letzte Änderung vorgenommen haben, und überprüfen den Quelltext. Interne Fehler sind oft durch wenige Zeilen reproduzierbar und werden meist durch ungewöhnliche oder unerwartete Konstrukte verursacht. Versuchen Sie in diesem Fall, die Programmierabsicht mit anderen Mitteln umzusetzen. Wenn Sie beispielsweise einen Wert in einen anderen Typ umwandeln, deklarieren Sie eine Variable des Zieltyps, und führen Sie zuerst eine Zuweisung aus.

```
begin
  if Integer(b) = 100 then...
end;
var
  a: Integer;
begin
  a := b;
  if a = 100 then...
end;
```

Hier ein Beispiel für unerwarteten Code, der korrigiert werden kann, um den Fehler zu beheben:

```
var
  A: Integer;
begin
  { Die folgende zweite Typumwandlung von A in Int64 ist unnötig. Wenn diese Operation entfernt wird, kann der interne Fehler vermieden werden. }
  if Int64(Int64(A))=0 then
end;
```

2. In diesem Fall ist die zweite Typumwandlung von `A` in `Int64` unnötig. Wenn diese Operation entfernt wird, ist der interne Fehler vermieden. Wenn das Problem in einer `while...do`-Schleife zu liegen scheint, versuchen Sie stattdessen eine

for...do-Schleife. Obwohl diese Maßnahme nicht das eigentliche Problem behebt, sollten Sie danach wenigstens in der Lage sein, an Ihrer Anwendung weiterzuarbeiten. Wenn sich das Problem mit einer solchen Maßnahme umgehen lässt, bedeutet dies nicht, dass die while- oder for-Schleife fehlerhaft war, sondern dass die Art und Weise der Codierung unerwartet war.

3. Wenn Sie das Problem eingrenzen konnten, bitten wir Sie, das kleinstmögliche Konstrukt zu extrahieren, an dem der Fehler zu beobachten ist, und dieses an Borland zu übermitteln.

Weitere Verfahren zur Beseitigung von internen Fehlern

1. Wenn das Problem in einer while...do-Schleife zu liegen scheint, versuchen Sie stattdessen eine for...do-Schleife und umgekehrt.
2. Wenn eine geschachtelte Funktion oder Prozedur die Problemstelle zu sein scheint, lösen Sie probehalber die Schachtelung auf.
3. Falls der Fehler sich in einer Typumwandlung befindet, suchen Sie Alternativen für die Typumwandlung, wie z.B. lokale Variablen des erforderlichen Typs.
4. Wenn das Problem in einer with-Anweisung auftritt, entfernen Sie die gesamte with-Anweisung.
5. Deaktivieren Sie probehalber die Compileroptimierungen unter [Projekt>Optionen>Compiler](#).

Wenn alle anderen Mittel versagen ...

1. Natürlich gibt es viele verschiedene Wege, Programmcode zu schreiben. Deshalb kann ein universeller Lösungsansatz für interne Fehler darin bestehen, den kritischen Code umzuschreiben. Dies ist vielleicht nicht immer die schnellste Lösung, ermöglicht es aber zumindest, die Arbeit an der Anwendung fortzusetzen. Wenn sich das Problem mit einer solchen Maßnahme umgehen lässt, bedeutet dies nicht, dass die while- oder for-Schleife fehlerhaft war, sondern dass die Art und Weise der Codierung unerwartet war und deshalb einen Fehler ausgelöst hat.
2. Wenn Sie den Code mit der neuesten Compiler-Version ausprobiert haben und der Fehler weiterbesteht, erstellen Sie den kleinstmöglichen Testfall, der den Fehler produziert, und senden Sie ihn an CodeGear. Falls der Fehler mit der neuesten Compiler-Version nicht auftritt, wurde das Problem wahrscheinlich bereits mit dem neuen Compiler behoben.

Die IDE zur Vermeidung von internen Fehlern konfigurieren

1. Legen Sie ein Verzeichnis an, in dem Sie alle .dcpil-Dateien (vorcompilierte Package-Dateien) ablegen. Erstellen Sie beispielsweise das Verzeichnis [C:\DCPIL](#), und öffnen Sie unter [Tools>Umgebungsoptionen](#) die Registerkarte Bibliothek. Setzen Sie dort das DCPIL-Ausgabeverzeichnis auf [C:\DCPIL](#). Dadurch ist sichergestellt, dass die vom Compiler erzeugten .dcpil-Dateien immer aktuell sind. Diese Maßnahme ist sinnvoll, wenn Sie ein Package in ein anderes Verzeichnis verschieben. Mit dem Befehl [Projekt>Optionen>Verzeichnisse/Bedingungen>Unit](#) können Sie ein projektbezogenes Ausgabeverzeichnis für die .dcu11-Dateien erstellen.
2. Es ist sehr wichtig, dass Sie immer die aktuellsten Versionen der .dcu11- und .dcpil-Dateien verwenden. Andernfalls können interne Fehler auftreten, die sich leicht vermeiden lassen.

Siehe auch

[Laufzeitfehler](#) (siehe Seite 1079)

[Schwerwiegende Fehler](#) (siehe Seite 1081)

[E/A-Fehler](#) (siehe Seite 1083)

[Betriebssystemfehler](#) (siehe Seite 1083)

[Liste aller Delphi-Compiler-Fehler und -Meldungen](#) (siehe Seite 1084)

4.1.3.3 Schwerwiegende Fehler

Diese Fehler führen in jedem Fall zu einer unmittelbaren Beendigung des Programms.

Exception-Zuordnung

In Anwendungen, die die Klasse SysUtils verwenden (wie die meisten GUI-Anwendungen), sind diesen Fehlern Exceptions zugeordnet. Eine Beschreibung der jeweiligen Fehlerbedingung finden Sie in der Dokumentation zur Exception.

E/A-Fehler

In der folgenden Tabelle sind alle schwerwiegenden Fehler einschließlich Nummer und zugehöriger Exception aufgeführt.

Nummer	Name	Exception
200	Division durch Null	EDivByZero
201	Fehler bei Bereichsprüfung	ERangeError
202	Stack-Überlauf	EStackOverflow
203	Heap-Überlauf	EOutOfMemory
204	Ungültige Zeigeroperation	EInvalidPointer
205	Gleitkommaüberlauf	EOverflow
206	Gleitkommaunterlauf	EUnderflow
207	Ungültige Gleitkommaoperation	EInvalidOp
210	Fehler bei abstrakter Methode	EAbstractError
215	Arithmetischer Überlauf (nur Integer)	EIntOverflow
216	Zugriffsverletzung	EAccessViolation
217	Control-C	EControlC
218	Privilegierte Anweisung	EPrivilege
219	Unzulässige Typkonvertierung	EInvalidCast
220	Unzulässige Variant-Typkonvertierung	EVariantError
221	Unzulässige Variant-Operation	EVariantError
222	Kein Aufruf-Dispatcher für variante Methode	EVariantError
223	Variantes Array kann nicht erzeugt werden	EVariantError
224	Variante enthält kein Array	EVariantError
225	Abgrenzungsfehler bei variantem Array	EVariantError
226	TLS-Initialisierungsfehler	Keine Zuordnung zu Exception möglich
227	Assertion fehlerhaft	EAssertionFailed
228	Interface-Umwandlungsfehler	EIntfCastError
229	Safecall-Fehler	ESafecallException
230	Unbehandelte Exception	Keine Zuordnung zu Exception möglich
231	Zu viele verschachtelte Exceptions	Maximal 16 sind zulässig
232	Signal für schwerwiegenden Fehler bei Nicht-Delphi-Thread	Keine Zuordnung zu Exception möglich

Siehe auch

Exceptions (siehe Seite 977)

Interne Fehler beheben (siehe Seite 1080)

Laufzeitfehler (siehe Seite 1079)

E/A-Fehler (siehe Seite 1083)

Betriebssystemfehler ([siehe Seite 1083](#))

4.1.3.4 E/A-Fehler

E/A-Fehler lösen eine Exception aus, wenn eine Anweisung im Status {\$I+} compiliert wird. Wird die Klasse SysUtils in der Anwendung nicht verwendet, führt die Exception zur Programmbeendigung.

Behandlung von E/A-Fehlern

Im Status {\$I-} wird die Programmausführung fortgesetzt, und der Fehler wird von der Funktion IOResult gemeldet.

E/A-Fehler

Die folgende Tabelle enthält alle E/A-Fehler mit zugehöriger Nummer und einer kurzen Beschreibung.

Nummer	Name	Beschreibung
100	Lesefehler auf Laufwerk	Wird von Read bei dem Versuch gemeldet, den Lesevorgang für eine Textdatei nach dem Dateiende fortzusetzen.
101	Schreibfehler auf Laufwerk	Wird von CloseFile, Write, WriteLn oder Flush gemeldet, wenn auf dem Laufwerk kein freier Speicher mehr verfügbar ist.
102	Datei nicht zugewiesen	Wird von Reset, Rewrite, Append, Rename und Erase gemeldet, wenn der Dateivariablen kein Name zugewiesen wurde (durch einen Aufruf von Assign oder AssignFile).
103	Datei nicht geöffnet	Wird von CloseFile, Read Write, Seek, Eof, FilePos, FileSize, Flush, BlockRead oder BlockWrite gemeldet, wenn die Datei nicht geöffnet ist.
104	Datei nicht für Eingabe geöffnet	Wird von Read, ReadLn, Eoln, Eoln, SeekEof oder SeekEoln gemeldet, wenn die Textdatei nicht für die Eingabe geöffnet ist.
105	Datei nicht für Ausgabe geöffnet	Wird von Write oder Writeln für eine Textdatei gemeldet, wenn keine Konsolenanwendung bereitgestellt wird.
106	Ungültiges Zahlenformat	Wird von Read oder ReadLn gemeldet, wenn ein aus einer Textdatei gelesener numerischer Wert nicht das richtige Zahlenformat hat.

Siehe auch

Exceptions ([siehe Seite 977](#))

Interne Fehler beheben ([siehe Seite 1080](#))

Laufzeitfehler ([siehe Seite 1079](#))

Schwerwiegende Fehler ([siehe Seite 1081](#))

Betriebssystemfehler ([siehe Seite 1083](#))

4.1.3.5 Betriebssystemfehler

Alle Fehler, die keine E/A- oder schwerwiegenden Fehler sind, werden mit den vom Betriebssystem zurückgegebenen Fehlercodes gemeldet.

Fehlercodes des Betriebssystems

Fehlercodes des Betriebssystems sind Rückgabewerte von Aufrufen für Betriebssystemfunktionen. Der Code für den zuletzt aufgetretenen Fehler kann mit der globalen Funktion GetLastError ermittelt werden. Soll für den letzten fehlgeschlagenen API-Aufruf eine Exception ausgelöst werden, rufen Sie stattdessen die Prozedur RaiseLastOSError auf.

Die von GetLastError zurückgegebenen Fehlercodes variieren je nach Betriebssystem. Den Fehler-String für einen Fehlercode

erhalten Sie durch einen Aufruf der globalen Funktion SysErrorMessage.

Rückgabewerte abrufen

Mit der globalen Funktion Win32Check können Sie den Rückgabewert für einen Aufruf einer Win32-API-Funktion ermitteln und, falls ein Fehler vorliegt, eine EWin32Error-Exception auslösen.

Siehe auch

Exceptions (siehe Seite 977)

Interne Fehler beheben (siehe Seite 1080)

Laufzeitfehler (siehe Seite 1079)

Schwerwiegende Fehler (siehe Seite 1081)

E/A-Fehler (siehe Seite 1083)

4.1.3.6 Fehlermeldungen

Dieser Abschnitt enthält die Delphi-Compiler-Fehlermeldungen und Warnungen von RAD Studio.

4.1.3.6.1 Thread-lokale Variablen können nicht ABSOLUTE sein (E2190)

Thread-lokale Variablen können sich weder auf eine andere Variable beziehen noch auf eine absolute Speicheradresse.

```
program Produce;
```

```
  threadvar
    secretNum : integer absolute $151;
```

```
begin
end.
```

Die Anweisung absolute ist im Abschnitt der threadvar-Deklaration nicht zulässig.

```
program Solve;
```

```
  threadvar
    secretNum : integer;
  var
    sNum : integer absolute $151;
```

```
begin
end.
```

Zur Lösung eines Problems dieser Art gibt es zwei einfache Möglichkeiten. Zum einen kann die Anweisung absolute aus dem Abschnitt threadvar entfernt werden. Zum anderen kann die absolute Variable in einen normalen Abschnitt der var-Deklaration gebracht werden.

4.1.3.6.2 Bei der Compilierung zu Byte-Code können absolute Variablen nicht benutzt werden (E2249)

Die Verwendung von absoluten Variablen ist beim Compilieren zum Erzeugen von Byte-Code nicht zulässig.

4.1.3.6.3 Aufruf der abstrakten Methode %s.%s (E2373)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.4 ABSTRACT und FINAL dürfen nicht zusammen verwendet werden (E2371)

Eine Klasse kann nicht gleichzeitig final und abstract sein.

Final ist ein restriktiver Modifizierer, der die Erweiterung von Klassen verhindert (oder das Überschreiben von Methoden verhindert), während der Modifizierer abstract anzeigt, dass eine Klasse als Basisklasse verwendet werden soll.

4.1.3.6.5 Definition für abstrakte Methode '%s' nicht erlaubt (E2136)

<Element> ist als abstract deklariert. Der Compiler hat jedoch in der Quelldatei eine Definition für die Methode gefunden. Es ist nicht erlaubt, für eine abstrakte Deklaration eine Definition bereitzustellen.

```
program Produce;

type
  Base = class
    procedure Foundation; virtual; abstract;
  end;

  procedure Base.Foundation;
  begin
  end;

begin
end.
```

Abstrakte Methoden dürfen nicht definiert werden. Wenn Sie dieses Programm compilieren, tritt an der Stelle Base.Foundation ein Fehler auf.

```
program Solve;

type
  Base = class
    procedure Foundation; virtual; abstract;
  end;

  Derived = class (Base)
    procedure Foundation; override;
  end;

  procedure Derived.Foundation;
  begin
  end;

begin
end.
```

Zur Behebung dieses Problems sind zwei Schritte erforderlich. Zuerst müssen Sie die Definition der abstrakten Prozedur in der Basisklasse entfernen. Dann müssen Sie die Basisklasse erweitern, die abstrakte Prozedur als override deklarieren und für die neue Prozedur eine Definition bereitstellen.

4.1.3.6.6 Abstrakte Methoden müssen virtuell oder dynamisch sein (E2167)

Wird eine abstrakte Methode in einer Basisklasse deklariert, muss sie entweder vom normalen virtuellen oder dynamischen virtuellen Typ sein.

```
program Produce;

type
  Base = class
    procedure DaliVision; abstract;
    procedure TellyVision; abstract;
  end;

begin
end.
```

Die Deklaration weist einen Fehler auf, da abstrakte Methoden entweder virtuell oder dynamisch sein müssen.

```
program Solve;

type
  Base = class
    procedure DaliVision; virtual; abstract;
    procedure TellyVision; dynamic; abstract;
  end;

begin
end.
```

Dieser Fehler kann entfernt werden, indem die Methode als virtual oder dynamic festgelegt wird, je nachdem, welcher Typ für Ihre Anwendung am besten geeignet ist.

4.1.3.6.7 ABSTRACT und SEALED dürfen nicht zusammen verwendet werden (E2383)

Eine Klasse kann nicht gleichzeitig sealed und abstract sein.

Der Modifizierer sealed verhindert die Vererbung von Klassen, während der Modifizierer abstract anzeigt, dass eine Klasse als Basisklasse verwendet werden soll.

4.1.3.6.8 Eine Adresse kann bei der Compilierung zu Byte-Code nicht akzeptiert werden (E2247)

Der Adressen-Operator @ kann bei der Compilierung zum Erzeugen von Byte-Code nicht verwendet werden.

4.1.3.6.9 Doppeldeutiger überladener Aufruf von '%s' (E2251)

Mit dem aktuellen Funktions- oder Prozedurenaufruf und der Überladungsliste für die angegebene Funktion kann der Compiler nicht ermitteln, welche Version der Prozedur aufgerufen werden soll.

```
program Produce;

procedure f0(a : integer); overload;
begin
end;

procedure f0(a : integer; b : char = 'A'); overload;
begin
```

```
end;  
  
begin  
  f0(1);  
end.
```

In diesem Beispiel macht der in einer der Versionen von f0 angegebene Standardparameter dem Compiler die Ermittlung der aufzurufenden Prozedur unmöglich.

```
program Solve;  
  
procedure f0(a : integer); overload;  
begin  
end;  
  
procedure f0(a : integer; b : char); overload;  
begin  
end;  
  
begin  
  f0(1);  
end.
```

Eine Lösung besteht darin, den Standardparameterwert zu entfernen. Dies führt dazu, dass diejenige Funktion aufgerufen wird, die nur den Parameter Integer akzeptiert. Dies ist die einzige Möglichkeit zum Aufrufen der Einparameterfunktion.

4.1.3.6.10 Überlauf bei Konvertierung oder arithmetischer Operation (E2099)

Der Compiler hat in einem arithmetischen Ausdruck einen Überlauf entdeckt. Das Ergebnis des Ausdrucks kann wegen seiner Größe nicht in 32 Bits dargestellt werden.

Überprüfen Sie die durchgeführten Berechnungen, und stellen Sie sicher, dass die Ergebnisse von der Hardware des Computers dargestellt werden können.

4.1.3.6.11 Standardfunktion NEW erwartet einen dynamischen Array-Typbezeichner (E2307)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.12 Es muss mindestens 1 Dimension für NEW des dyn. Array angegeben werden (E2308)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.13 Es muss mindestens 1 Dimension für SetLength des dyn. Array angegeben werden (E2246)

Die Standardprozedur SetLength wurde zur Änderung der Länge eines dynamischen Arrays aufgerufen, aber es wurden keine Array-Dimensionen angegeben.

```
program Produce;  
  
var  
  arr : array of integer;  
  
begin
```

```
  SetLength(arr);
end.
```

Dieser Aufruf von SetLength verursacht einen Fehler, weil keine Array-Dimensionen angegeben wurden.

```
program solve;

var
  arr : array of integer;

begin
  SetLength(arr, 151);
end.
```

Um diesen Fehler zu vermeiden, geben Sie die Anzahl der Elemente an, die das Array enthalten soll.

4.1.3.6.14 Zuweisung an FOR-Schleifenvariable '%s' (E2081)

Der Steuervariablen einer for-Schleife darf in der Schleife kein Wert zugewiesen werden.

Verwenden Sie die Anweisung break oder goto, um die Schleife vorzeitig zu verlassen.

```
program Produce;

var
  I: Integer;
  A: array[0..99] of Integer;
begin
  for I := 0 to 99 do begin
    if A[I] = 42 then
      I := 99;
  end;
end.
```

Hier sollte die Schleife durch Zuweisen des Wertes 99 beendet werden.

```
program Solve;

var
  I: Integer;
  A: array[0..99] of Integer;
begin
  for I := 0 to 99 do begin
    if A[I] = 42 then
      Break;
  end;
end.
```

Besser ist es, die Schleife mit einer break-Anweisung zu verlassen.

4.1.3.6.15 Zuweisung für typisierte Konstante '%s' (W1017)

Diese Warnmeldung wird momentan nicht verwendet.

4.1.3.6.16 486/487-Instruktionen sind nicht aktiviert (E2117)

Dieser Fehler sollte nicht auftreten, da 486-Instruktionen immer aktiviert sind.

4.1.3.6.17 Ungültige Kombination von Opcode und Operanden (E2116)

Sie haben eine ungültige Anweisung für den Inline Assembler angegeben.

```
program Produce;

procedure AssemblerExample;
asm
  mov al, $0f0 * 16
end;
```

```
begin
end.
```

Der Inline-Assembler kann das Ergebnis von $\$f0 * 16$ nicht im 'al'-Register ablegen (unpassende Größe).

```
program Solve;
procedure AssemblerExample;
asm
  mov al, $0f * 16
end;
```

```
begin
end.
```

Stellen Sie sicher, dass die Typen der Operanden zueinander kompatibel sind.

4.1.3.6.18 Konstante erwartet (E2109)

Der Inline Assembler erwartet eine Konstante, konnte jedoch keine finden.

```
program Produce;

procedure Assembly(x : Integer);
asm
  mov ax, x MOD 10
end;
```

```
begin
end.
```

Da der Inline Assembler mit einer Delphi-Variable keine MOD-Operation durchführen kann, ergibt der obige Quelltext einen Fehler.

Viele Ausdrücke des Inline Assemblers funktionieren nur mit Konstanten. Die fehlerhafte Anweisung muss entsprechend geändert werden.

4.1.3.6.19 Division durch Null (E2118)

Der Inline Assembler hat einen Ausdruck entdeckt, der eine Division durch Null ergibt.

```
program Produce;

procedure AssemblerExample;
asm
  dw 1000 / 0
end;
```

```
begin
end.
```

Wenn Sie Konstanten anstelle von konstanten Literalen verwenden, ist der Fehler vielleicht nicht so offensichtlich.

```
program Solve;

procedure AssemblerExample;
asm
  dw 1000 / 10
```

```
end;
```

```
begin
end.
```

Stellen Sie sicher, dass in Ihren Programmen nicht durch 0 dividiert wird.

4.1.3.6.20 Strukturfeldbezeichner erwartet (E2119)

Ein Bezeichner auf der rechten Seite des Punktes ist kein Feld des Records, der links vom Punkt angegeben ist. Eine mögliche und schwer zu findende Ursache für diesen Fehler ist, dass Sie in einem Record den Feldbezeichner Ch verwenden, der vom Inline Assembler als Registername interpretiert wird.

```
program Produce;

type
  Data = record
    x: Integer;
  end;

procedure AssemblerExample(d : Data; y : Char);
asm
  mov  eax, d.y
end;

begin
end.
```

In diesem Beispiel erkennt der Inline Assembler y als gültigen Bezeichner, aber nicht als Element von d.

```
program Solve;

type
  Data = record
    x: Integer;
  end;

procedure AssemblerExample(d : Data; y : Char);
asm
  mov  eax, d.x
end;

begin
end.
```

Geben Sie den richtigen Variablenamen an.

4.1.3.6.21 Speicherreferenz erwartet (E2108)

Der Inline Assembler erwartet eine Speicherreferenz, konnte jedoch keine finden.

Stellen Sie sicher, dass es sich bei der Anweisung, die den Fehler verursacht, tatsächlich um eine Speicherreferenz handelt.

4.1.3.6.22 Fehler in numerischer Konstante (E2115)

Der Inline Assembler hat in einer numerischen Konstanten einen Fehler entdeckt.

```
program Produce;

procedure AssemblerExample;
asm
  mov al, $z0f0
```

```
end;
```

```
begin
end.
```

In diesem Beispiel findet der Inline Assembler anstelle einer hexadezimalen Konstanten ein fehlerhaftes Zeichen.

```
program Solve;

procedure AssemblerExample;
asm
  mov al, $f0
end;
```

```
begin
end.
```

Stellen Sie sicher, dass numerische Konstanten dem Typ entsprechen, den der Inline Assembler erwartet.

4.1.3.6.23 Operandengröße stimmt nicht überein (E2107)

Die vom Operanden benötigte Größe stimmt nicht mit der angegebenen Größe überein.

```
program Produce;

var
  v : Integer;

procedure Assembly;
asm
  db offset v
end;
```

```
begin
end.
```

In diesem Beispiel gibt der Compiler einen Fehler aus, da der Operator offset einen dword-Wert generiert, aber ein byte-Wert erwartet wird.

```
program Solve;

var
  v : Integer;

procedure Assembly;
asm
  dd offset v
end;
```

```
begin
end.
```

Die Lösung für dieses Beispiel besteht darin, den Operator für die Übernahme eines dword-Werts einzurichten. Überprüfen Sie Ihren Quelltext genau, und stellen Sie sicher, dass die Größe des Operators und des Operanden übereinstimmen.

4.1.3.6.24 Numerischer Überlauf (E2113)

Der Inline-Assembler hat in einem Ausdruck einen numerischen Überlauf entdeckt.

```
program Produce;

procedure AssemblerExample;
asm
  mov eax, $0xffffffffffffffffffff
```

```
end;
```

```
begin
end.
```

Dieser Fehler tritt auf, wenn eine Zahl nicht mit 32 Bits dargestellt werden kann.

```
program Solve;
```

```
procedure AssemblerExample;
asm
  mov al, $0ff
end;
```

```
begin
end.
```

Stellen Sie sicher, dass alle Zahlen mit 32 Bits darstellbar sind.

4.1.3.6.25 Ungültige Registerkombination (E2112)

Sie haben in einer Anweisung für den Inline Assembler eine ungültige Registerkombination angegeben. Weitere Informationen über die Adressierungsmodi der Familie Intel 80x86 finden Sie in einem Assembler-Handbuch.

```
program Produce;
```

```
procedure AssemblerExample;
asm
  mov eax, [ecx + esp * 4]
end;
```

```
begin
end.
```

Der rechte Operand dieser mov-Anweisung ist ungültig.

```
program Solve;
```

```
procedure AssemblerExample;
asm
  mov eax, [ecx + ebx * 4]
end;
```

```
begin
end.
```

Der Adressierungsmodus des rechten Operanden dieser mov-Anweisung ist gültig.

4.1.3.6.26 Relozierbare Symbole dürfen nicht addiert oder subtrahiert werden (E2111)

Der Inline Assembler kann keine Speicheradressen addieren oder subtrahieren, die möglicherweise vom Linker geändert werden.

```
program Produce;
```

```
var
  a: array[1..0,10] of Integer;
  endOfA : Integer;

procedure Relocatable;
begin
end;
```

```
procedure Assembly;
asm
  mov eax, a + endOfA
end;

begin
end.
```

Globale Variablen gehören zu den Elementen, die relozierbare Adressen produzieren. Der Inline Assembler kann solche Adressen weder addieren noch subtrahieren.

Stellen Sie sicher, dass in Ihren Anweisungen für den Inline Assembler keine relozierbaren Adressen addiert oder subtrahiert werden.

4.1.3.6.27 Inline Assembler Stack-Überlauf (E2106)

Der angegebene Code überschreitet die Kapazität des Inline Assemblers.

Wenden Sie sich an CodeGear, wenn Sie diese Fehlermeldung erhalten.

4.1.3.6.28 Stringkonstante zu lang (E2114)

Der Inline Assembler hat das Ende eines Strings nicht gefunden. Der häufigste Grund für diesen Fehler ist ein fehlendes Anführungszeichen.

```
program Produce;

procedure AssemblerExample;
asm
  db 'Hello world. Ich bin eine Inline-Assembler-Anweisung
end;

begin
end.
```

Da der Inline Assembler das Ende des Strings nicht finden kann, gibt er einen Fehler aus.

```
program Solve;

procedure AssemblerExample;
asm
  db 'Hello world. Ich bin eine Inline-Assembler-Anweisung'
end;
```

```
begin
end.
```

Das schließende Anführungszeichen behebt das Problem.

4.1.3.6.29 Inline Assembler Syntaxfehler (E2105)

Sie haben einen Ausdruck angegeben, den der Inline Assembler nicht als gültige Anweisung interpretieren kann.

```
program Produce;

procedure Assembly;
asm
  adx eax, 151
end;

begin
```

```
end.  
program Solve;  
  
procedure Assembly;  
asm  
  add  eax, 151  
end;
```

```
begin  
end.
```

Überprüfen Sie die Anweisung, die den Fehler verursacht, und stellen Sie sicher, dass sie der richtigen Syntax entspricht.

4

4.1.3.6.30 Typ erwartet (E2110)

Wenden Sie sich an CodeGear, wenn Sie diesen Fehler erhalten.

4.1.3.6.31 Ein Attribut-Argument muss ein konstanter Ausdruck, ein typeof-Ausdruck oder ein Array-Konstruktor sein (E2448)

Die CLR (Common Language Runtime) legt fest, dass ein Attribut-Argument ein konstanter Ausdruck, ein `typeof`-Ausdruck oder ein Array-Konstruktor sein muss. Attribut-Argumente dürfen z.B. keine globalen Variablen sein. Attribut-Instanzen werden zur Compilierzeit erstellt und in die Assemblierungs-Metadaten eingefügt, daher können Sie nicht von Laufzeitinformationen erzeugt werden.

4.1.3.6.32 Falsches Objektdateiformat: '%s' (E2045)

Diese Warnung wird angezeigt, wenn eine mit der Direktive `$L` oder `$LINK` eingebundene Objektdatei nicht das richtige Format hat. Es müssen folgende Beschränkungen eingehalten werden:

- Überprüfen Sie die Namensbeschränkungen bei Segmentdateien in der Hilfdatei.
- Es dürfen nicht mehr als 10 Segmente vorhanden sein.
- Es dürfen nicht mehr als 255 externe Symbole definiert sein.
- L NAMES-Records dürfen nicht mehr als 50 lokale Namen enthalten.
- LEDATA- und LDATA-Records müssen die Offset-Reihenfolge einhalten.
- In FIXU32-Records werden keine THREAD-Subrecords unterstützt.
- Fixups sind nur bei 32-Bit-Offsets möglich.
- Es sind nur Segment- und selbstbezügliche Fixups möglich.
- Das Ziel eines Fixup muss ein Segment, eine Gruppe oder ein EXTDEF sein.
- Objekt muss eine 32-Bit-Objektdateien ein.
- Es werden verschiedene interne Konsistenzprüfungen durchgeführt, die nur bei einer beschädigten Objektdatei fehlschlagen können.

4.1.3.6.33 Falsche globale Symboldefinition: '%s' in Objektdatei '%s' (x1028)

Diese Warnung wird angezeigt, wenn eine mit der Direktive `$L` oder `$LINK` eingebundene Objektdatei die Definition eines Symbols enthält, das in Delphi nicht als externe Prozedur deklariert wurde, sondern als etwas anderes (z. B. als Variable).

Die Symboldefinition wird in diesem Fall ignoriert.

4.1.3.6.34 Typ in einem OLE-Automatisierungsaufaufruf nicht erlaubt (E2160)

Ein Datentyp, der vom Compiler nicht in einen Variant-Typ konvertiert werden kann, ist in einem Aufruf der OLE-Automatisierung nicht zulässig.

```
program Produce;

type
  Base = class
    x: Integer;
  end;

var
  B : Base;
  V : Variant;

begin
  V.Dispatch(B);
end.
```

Eine Klasse kann nicht in einen Variant-Typ konvertiert werden und ist daher in einem OLE-Aufruf nicht zulässig.

```
program Solve;

type
  Base = class
    x: Integer;
  end;

var
  B : Base;
  V : Variant;

begin
  V.Dispatch(B);
end.
```

Die einzige Lösung für dieses Problem ist die manuelle Konvertierung dieser Datentypen in den Typ Variant bzw. die ausschließliche Verwendung von Datentypen, die automatisch in den Typ Variant konvertiert werden können.

4.1.3.6.35 Published-Eigenschaft '%s' kann nicht vom Typ %s sein (E2188)

published-Eigenschaften müssen ein Ordinaltyp, Single, Double, Extended oder Comp, ein Stringtyp, ein Mengentyp (der mit 32 Bit auskommt) oder ein Methoden-Zeigertyp sein. Wenn der Compiler in einem öffentlichen Abschnitt auf einen anderen Eigenschaftentyp trifft, entfernt er das Attribut published.

```
(*$TYPEINFO ON*)
program Produce;

type
  TitleArr = array [0..24] of char;
  NamePlate = class
  private
    titleStr : TitleArr;
  published
    property Title: TitleArr read titleStr write titleStr;
  end;

begin
end.
```

Hier wird ein Fehler ausgelöst, da ein Array nicht einer der Datentypen ist, die veröffentlicht werden können.

```
(*$TYPEINFO ON*)
program Solve;

type
  TitleArr = integer;
  NamePlate = class
    titleStr : TitleArr;
  published
    property Title: TitleArr read titleStr write titleStr;
  end;

begin
end.
```

Dieser Fehler wird vermieden, indem die Deklaration der Eigenschaft aus dem öffentlichen Abschnitt entfernt wird. Eine andere Möglichkeit, wie in diesem Beispiel verwirklicht, liegt darin, den Typ der Eigenschaft in einen zu verändern, der veröffentlicht werden kann.

4.1.3.6.36 Ungültiger Typ in Read/Readln-Anweisung (E2055)

Sie haben versucht, eine Variable eines unzulässigen Typs in einer Read- oder Readln-Anweisung zu lesen.

Überprüfen Sie den Variabtentyp, und vergewissern Sie sich, dass kein Dereferenzierungs-, Indizierungs- oder Feldauswahloperator fehlt.

```
program Produce;
type
  TColor = (red,green,blue);
var
  Color : TColor;
begin
  Readln(Color);      (*<-- Hier die Fehlermeldung*)
end.
```

Variablen eines Aufzählungstyps können nicht direkt gelesen werden.

```
program Solve;
type
  TColor = (red,green,blue);
var
  Color : TColor;
  InputString: string;
const
  ColorString : array [TColor] of string = ('red', 'green', 'blue');
begin
  Readln(InputString);
  Color := red;
  while (color < blue) and (ColorString[color] <> InputString) do
    Inc(color);
end.
```

Die Lösung besteht darin, einen String einzulesen und diesen anschließend in einer Hilfstabelle zu suchen. Im obigen Beispiel wurde keine Fehlerprüfung durchgeführt, jeder String wird als blue behandelt. In einer Anwendung würden Sie eine Fehlermeldung ausgeben und den Benutzer zur erneuten Eingabe auffordern.

4.1.3.6.37 Syntaxfehler in Real-Zahl (E2053)

Diese Fehlermeldung wird angezeigt, wenn der Compiler den Beginn eines Skalierungsfaktors (das Zeichen "E" oder "e") in einer Zahl findet, aber keine nachfolgenden Ziffern.

```
program Produce;
const
```

```
SpeedOfLight = 3.0E 8;      (*<-- Hier die Fehlermeldung*)
begin
end.
```

In diesem Beispiel folgt nach "3.0E" ein Leerzeichen. Für den Compiler ist damit die Zahl zu Ende und somit unvollständig.

```
program Solve;
const
  SpeedOfLight = 3.0E+8;
begin
end.
```

Wir hätten einfach das Leerzeichen löschen können, haben aber das Vorzeichen "+" eingegeben, weil es schöner aussieht.

4.1.3.6.38 Falsche Adressverschiebung in Objektdatei '%s' entdeckt (E2104)

Sie versuchen, mit der Compiler-Direktive \$L Objektmodule zu Ihrem Programm zu linken. Der Compiler kann die Objektdatei jedoch nicht verarbeiten; sie ist zu komplex. Die Einbindung von C++ Objektdateien wird beispielsweise nicht unterstützt.

4.1.3.6.39 Unit %s veraltet oder beschädigt: '%s' fehlt (E2158)

Der Compiler sucht eine spezielle Funktion, die sich in System.dcu befindet, kann sie jedoch nicht finden. Ihre System-Unit ist beschädigt oder veraltet.

Stellen Sie sicher, dass in Ihrem Suchpfad für Bibliotheken keine Konflikte vorhanden sind, so dass der Pfad auf eine andere System.dcu verweist. Installieren Sie die Datei System.dcu neu. Wenn dies nicht zum Erfolg führt, wenden Sie sich an den CodeGear-Entwickler-Support.

4.1.3.6.40 Unit %s veraltet oder beschädigt: '%s.%s' fehlt (E2159)

Der Compiler hat eine bestimmte Funktion in der Unit System nicht gefunden, das bedeutet, dass die im Suchpfad gefundene Unit entweder beschädigt oder veraltet ist.

4.1.3.6.41 Typ ist im Aufruf von Variant Dispatch nicht zulässig (E2281)

Sie haben versucht, eine Methode mit einem Argument aufzurufen, für dessen Typ der Compiler kein Marshalling durchführen kann. Variant-Typen können Interfaces aufnehmen, aber die Interfaces können nur für bestimmte Typen ein Marshalling durchführen.

Unter Windows unterstützt Delphi COM- und SOAP-Interfaces und kann Typen aufrufen, für die von diesen Interfaces ein Marshalling durchgeführt werden kann.

4.1.3.6.42 Falscher Argumententyp im Konstruktor des VariablenTyp array (E2150)

Sie versuchen, ein Array mithilfe eines Typs zu konstruieren, der für Variablenarrays nicht zulässig ist.

```
program Produce;
type
  Fruit = (apple, orange, pear);
  Data = record
    x: Integer;
    ch : Char;
  end;
```

```

var
  f : Fruit;
  d : Data;

procedure Examiner(v : array of TVarRec);
begin
end;

begin
  Examiner([d]);
  Examiner([f]);
end.

```

Beide Aufrufe von Examiner werden fehlschlagen, weil Aufzählungen und Records in Array-Konstruktoren nicht unterstützt werden.

```

program Solve;

var
  i : Integer;
  r : Real;
  v : Variant;

procedure Examiner(v : array of TVarRec);
begin
end;

begin
  i := 0; r := 0; v := 0;
  Examiner([i, r, v]);
end.

```

In Array-Konstruktoren sind viele Datentypen zulässig, unter anderem die in diesem Beispiel.

4.1.3.6.43 Ungültiger Typ in Write/Writeln-Anweisung (E2054)

Sie haben versucht, einen unzulässigen Typ in einer Write- oder Writeln-Anweisung auszugeben.

```

program Produce;
type
  TColor = (red,green,blue);
var
  Color : TColor;
begin
  Writeln(Color);
end.

```

Wäre es nicht bequem, Color mit einer Writeln-Anweisung auszugeben?

```

program Solve;
type
  TColor = (red,green,blue);
var
  Color : TColor;
const
  ColorString : array [TColor] of string = ('red', 'green', 'blue');
begin
  Writeln(ColorString[Color]);
end.

```

Leider ist das nicht zulässig, und wir müssen uns mit einer Hilfstabelle behelfen.

4.1.3.6.44 Prozedurendefinition muss der ILCODE-Aufrufkonvention entsprechen (E2297)

Verwaltetet .NET-Quelltext darf nur ILCODE-Aufrufkonvention verwenden

4.1.3.6.45 Anweisungen sind im Interface-Teil nicht erlaubt (E2050)

Der interface-Abschnitt einer Unit darf nur Deklarationen enthalten, aber keine Anweisungen.

Verschieben Sie den Prozedurrumpf in den implementation-Abschnitt.

```
unit Produce;

interface

procedure MyProc;           (*<-- Hier die Fehlermeldung*)
begin
end;

implementation

begin
end.
```

Das Problem tritt auf, weil sich der Prozedurrumpf im interface-Abschnitt befindet.

```
unit Solve;

interface

procedure MyProc;
implementation

procedure MyProc;
begin
end;

begin
end.
```

Die Prozedur muss in den implementation-Abschnitt verschoben werden.

4.1.3.6.46 Konstantenausdruck verletzt untere Grenzen (x1012)

Diese Fehlermeldung tritt auf, wenn der Compiler feststellt, dass eine Konstante außerhalb der zulässigen Grenzen liegt. Dies kann beispielsweise dann eintreten, wenn Sie eine Konstante einer unterdimensionierten Variablen zuweisen.

```
program Produce;
var
  Digit: 1..9;
begin
  Digit := 0;  (*Meldung: Konstantenausdruck verletzt untere Grenzen*)
end.

program Solve;
var
  Digit: 0..9;
begin
  Digit := 0;
end.
```

4.1.3.6.47 BREAK oder CONTINUE außerhalb der Schleife (E2097)

Der Compiler hat eine break- oder continue-Anweisung außerhalb einer while- oder repeat-Schleife entdeckt. Die beiden Konstrukte sind jedoch nur in Schleifen erlaubt.

```
program Produce;

procedure Error;
  var i: Integer;
begin
  i := 0;
  while i < 100 do
    INC(i);
    if odd(i) then begin
      INC(i);
    continue;
    end;
  end;

begin
end.
```

Die continue-Anweisung in diesem Beispiel befindet sich nur scheinbar, aber nicht wirklich innerhalb der while-Schleife.

```
program Solve;

procedure Error;
  var i: Integer;
begin
  i := 0;
  while i < 100 do begin
    INC(i);
    if odd(i) then begin
      INC(i);
    continue;
    end;
  end;
end;

begin
end.
```

Stellen Sie sicher, dass Ihre continue- und break-Anweisungen innerhalb der Schleife liegen, indem Sie Ihre Schleifen immer in Verbundanweisungen einschließen.

4.1.3.6.48 \$WEAKPACKAGEUNIT & \$DENYPACKAGEUNIT wurden beide angegeben (E2222)

Die gleichzeitige Angabe von \$WEAKPACKAGEUNIT und \$DENYPACKAGEUNIT ist nicht zulässig. Korrigieren Sie den Programmtext und führen Sie eine erneute Compilierung durch.

4.1.3.6.49 Bezeichner '%s' kann nicht exportiert werden (E2276)

Sie haben versucht, eine als local deklarierte Funktion oder Prozedur zu exportieren. Diese Fehlermeldung wird auch beim Exportieren von threadvars angezeigt.

4.1.3.6.50 Dieser Typ kann nicht initialisiert werden (E2071)

Dateitypen (einschließlich Textdateien) und Variant-Typen können nicht initialisiert werden. Sie können daher keine typisierten Konstanten oder initialisierte Variablen mit diesen Typen deklarieren.

```
program Produce;

var
  V : Variant = 0;

begin
end.
```

In diesem Beispiel wird eine initialisierte Variable des Typs Variante deklariert. Dies ist nicht zulässig.

```
program Solve;

var
  V : Variant;

begin
  V := 0;
end.
```

Die Lösung besteht darin, eine normale Variable mit einer Zuweisung zu initialisieren.

4.1.3.6.51 Aus %s kann kein eindeutiger Typ erstellt werden (E2374)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.52 \$DENYPACKAGEUNIT '%s' kann nicht in ein Package übernommen werden (E2223)

Sie haben versucht, eine mit \$DENYPACKAGEUNIT compilierte Unit in ein Package einzufügen. Dies ist nicht zulässig.

4.1.3.6.53 Das Published-Feld '%s' ist weder vom Typ class noch interface (E2217)

Es wurde versucht, ein Feld in einer Klasse als published zu deklarieren, das weder ein Klassen- noch ein Schnittstellentyp ist.

```
program Produce;

type
  TBaseClass = class
  published
    x: Integer;
  end;
begin
end.
```

Dieses Programm verursacht einen Fehler, weil 'x' in einem published-Abschnitt enthalten ist, obwohl 'x' kein Typ ist, der als published deklariert werden kann.

```
program Solve;
type
  TBaseClass = class
  published
    property X : Integer read Fx write Fx;
```

```
end;
```

```
begin
end.
```

Sie müssen alle Felder, die keine Klassen- oder Schnittstellentypen sind, aus dem published-Abschnitt einer Klasse löschen. Muss das Feld als published deklariert werden, ändern Sie es in eine Eigenschaft um, so wie es im Beispiel gezeigt wird.

4.1.3.6.54 Published-Methode '%s' enthält einen Typ, der nicht als published verwendet werden kann (E2218)

Diese Meldung wird in dccil nicht verwendet. Diese Meldung bezieht sich nur auf Win32-Compilierungen. Sie gibt an, dass der Ergebnistyp eines Parameters oder einer Funktion in der Methode kein Typ ist, der als published verwendet werden kann.

4.1.3.6.55 Adresse von lokalem Symbol %s kann nicht aufgenommen werden (E2278)

Sie haben versucht, ein Symbol in einer als local deklarierten Funktion oder Prozedur aufzurufen.

Mit local werden Routinen als nicht exportierbar gekennzeichnet. Dies ist plattformspezifisch und hat unter Windows keine Wirkung.

In Linux wird local für Routinen verwendet, die in eine Bibliothek kompiliert, aber nicht exportiert werden. Die Direktive kann nur für eigenständige Prozeduren und Funktionen, nicht aber für Methoden verwendet werden. Hier ein Beispiel für eine als local deklarierte Routine:

```
function Contraband(I: Integer): Integer; local;
```

aktualisiert z. B. nicht das Register EBX. Dies hat für die Routine folgende Auswirkungen:

- Sie kann nicht aus einer Bibliothek exportiert werden.
- Die Routine kann nicht im interface-Abschnitt einer Unit deklariert werden.
- Die Adresse der Routine kann nicht von einer Variable eines prozeduralen Typs übernommen oder einer solchen zugewiesen werden.
- Wenn es sich um eine reine Assembler-Routine handelt, kann sie nur dann aus einer anderen Unit aufgerufen werden, wenn der Aufrufer das EBX-Register aktualisiert.

4.1.3.6.56 Die Erzeugung der für die Eigenschaft %s.%s erforderliche(n) Zugriffsmethode(n) ist wegen eines Namenskonflikts mit einem vorhandenen Symbol in demselben Gültigkeitsbereich nicht möglich (E2392)

Für CLR ist erforderlich, dass für den Zugriff auf Eigenschaften Methoden und nicht Felder verwendet werden. In der Delphi-Sprache ist es möglich als Eigenschaftszugriffsmethoden Felder festzulegen. Der Delphi-Compiler erzeugt im Hintergrund die benötigten Methoden. CLS empfiehlt eine spezielle Namenskonvention für Eigenschaftszugriffsmethoden: `get_propname` and `set_propname`. Wenn nicht mit Methoden auf Eigenschaften zugegriffen wird oder die gegebene Methode nicht mit CLS-Namensmustern übereinstimmt; versucht der Delphi-Compiler, Methoden mit konformen CLS zu erzeugen. Ist in der Klasse, die mit den CLS-Namensmustern übereinstimmt, bereits eine Methode vorhanden, aber nicht der Eigenschaft zugeordnet, kann der Compiler keinen neuen Eigenschaftszugriffsmethode mit dem CLS-Namensmuster erstellen.

Wenn mit Methoden auf die Eigenschaft zugegriffen wird, verhindern Namenskollisionen, dass der Compiler einen CLS-konformen Namen erzeugt, verhindern aber nicht, dass die Eigenschaft verwendet werden kann.

Wenn jedoch wegen eines Namenskonflikts keine Eigenschaftszugriffsmethode für einen Feldzugriff vom Compiler erzeugt

werden kann, kann die Eigenschaft nicht verwendet werden und Sie erhalten diesen Fehler.

4.1.3.6.57 BREAK, CONTINUE oder EXIT bei FINALLY-Klausel nicht möglich (E2126)

Wenn ein Programm während der Exception-Behandlung in eine finally-Klausel eintritt, ist es nicht möglich, die Klausel mit break, continue oder exit zu verlassen, da die Steuerung an das Exception-Behandlungssystem zurückgegeben werden muss. Wenn in FINALLY durch den Exception-Behandlungsmechanismus eingetreten wird, kann die Klausel nicht mit BREAK, CONTINUE oder EXIT verlassen werden - wenn die finally-Klausel von dem Exception-Behandlungssystem ausgeführt wird, muss die Steuerung auch an das Exception-Behandlungssystem zurückgegeben werden.

```
program Produce;

procedure A0;
begin
  try
    (* etwas probieren, was fehlschlagen kann *)
  finally
    break;
  end;
end;

begin
end.
```

Das obige Programm versucht, die finally-Klausel mit einer break-Anweisung zu verlassen. Das ist nicht erlaubt.

```
program Solve;

procedure A0;
begin
  try
    (* etwas probieren, was fehlschlagen kann *)
  finally
  end;
end;

begin
end.
```

Die einzige Lösung für dieses Problem besteht darin, die Fehler verursachende Anweisung aus der finally-Klausel herauszunehmen.

4.1.3.6.58 Case-Label außerhalb des Bereichs des Case-Ausdrucks (W1018)

Sie haben innerhalb einer case-Anweisung ein Label angegeben, das von der Steuervariable der case-Anweisung nicht erzeugt werden kann.

```
program Produce;
(*$WARNINGS ON*)

type
  CompassPoints = (n, e, s, w, ne, se, sw, nw);
  FourPoints = n..w;

var
  TatesCompass : FourPoints;

begin
  TatesCompass := e;
```

```

  case TatesCompass OF
  n:   Writeln('North');
  e:   Writeln('East');
  s:   Writeln('West');
  w:   Writeln('South');
  ne:  Writeln('Northeast');
  se:  Writeln('Southeast');
  sw:  Writeln('Southwest');
  nw:  Writeln('Northwest');
  end;
end.

```

Ein TatesCompass kann nicht alle Werte der CompassPoints aufnehmen, und daher werden mehrere der Case-Labels Fehler auslösen.

```

program Solve;
(*$WARNINGS ON*)

type
  CompassPoints = (n, e, s, w, ne, se, sw, nw);
  FourPoints = n..w;

var
  TatesCompass : CompassPoints;

begin
  TatesCompass := e;
  case TatesCompass OF
  n:   Writeln('North');
  e:   Writeln('East');
  s:   Writeln('West');
  w:   Writeln('South');
  ne:  Writeln('Northeast');
  se:  Writeln('Southeast');
  sw:  Writeln('Southwest');
  nw:  Writeln('Northwest');
  end;
end.

```

Zur Behebung des Problems gibt es zwei Möglichkeiten: Zum einen kann die Steuervariable der Case-Anweisung geändert werden, damit sie alle Case-Labels erzeugen kann. Die zweite Möglichkeit wäre, alle Case-Labels zu entfernen, die die Steuervariable nicht erzeugen kann. In diesem Beispiel wird die erste Möglichkeit gezeigt.

4.1.3.6.59 Attribut '%s' kann nur einmal pro Ziel verwendet werden (E2326)

Dieses Attribut kann nur einmal pro Ziel verwendet werden. Attribute und deren Nachkommen können mit dem Attribut AttributeUsage deklariert sein, was beschreibt, wie ein benutzerdefiniertes Attribut eingesetzt wird. Wenn die Verwendung eines Attributs im Konflikt mit AttributeUsage.allowmultiple steht, wird dieser Fehler ausgelöst.

4.1.3.6.60 Attribut '%s' ist für dieses Ziel nicht gültig (E2325)

Das Attribut ist für dieses Ziel nicht gültig. Attribute und deren Nachkommen können mit dem Attribut AttributeUsage deklariert sein, was beschreibt, wie ein benutzerdefiniertes Attribut eingesetzt wird. Wenn die Verwendung eines Attributs im Konflikt mit AttributeUsage.validon steht, wird dieser Fehler ausgelöst. AttributeUsage.validon legt das Anwendungselement fest, für das dieses Attribut angewendet werden soll.

4.1.3.6.61 Attribut - Benanntes Argument eines bekannten Attributs darf kein Array sein (E2309)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.62 Attribut - Für einen benutzerdefinierten Marshaler ist ein benutzerdefinierter Marshaler-Typ erforderlich (E2310)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.63 Linker-Fehler bei der Ausgabe des Attributs '%s' (E2327)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.64 Attribut - Für den festen MarshalAs-String ist eine Größenangabe erforderlich (E2311)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.65 Attribut - Bekanntes Attribut darf keine Eigenschaften festlegen (E2313)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.66 Attribut - Ungültiges Argument für ein bekanntes Attribut (E2312)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.67 Attribut - Das Attribut MarshalAs hat Felder, die für den angegebenen unverwalteten Typ nicht zulässig sind (E2314)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.68 Attribut - Bekanntes benutzerdefiniertes Attribut für ungültiges Ziel (E2315)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.69 Attribut - Das Format der GUID war ungültig (E2316)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.70 Attribut - Bekanntes benutzerdefiniertes Attribut hatte ungültigen Wert (E2317)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.71 Attribut - Die Konstantengröße von MarshalAs darf nicht negativ sein (E2318)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.72 Attribut - Die Parametergröße von MarshalAs darf nicht negativ sein (E2319)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.73 Attribut - Der angegebene unverwaltete Typ ist nur für Felder gültig (E2320)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.74 Attribut - Bekanntes benutzerdefiniertes Attribut wiederholt das benannte Argument (E2321)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.75 Attribut - Unerwarteter Typ in bekanntem Attribut (E2322)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.76 Attribut - Unerwartetes Argument für ein bekanntes Attribut (E2323)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.77 Attribut - Benanntes Argument des bekannten Attributs unterstützt keine Varianten (E2324)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.78 Klassenkonstruktoren sind in unterstützenden Klassen nicht zulässig (E2358)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.79 Klassenkonstruktoren dürfen keine Parameter haben (E2360)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.80 Metadaten - Daten zu groß (E2340)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.81 Metadaten - Primärschlüsselspalte lässt möglicherweise keine Nullwerte zu (E2343)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.82 Metadaten - Spalte darf nicht geändert werden (E2341)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.83 Metadaten - Zu viele RID- oder Primärschlüsselspalten, max 1 (E2342)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.84 Metadaten - Fehler beim Lesen (E2329)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.85 Metadaten - Fehler beim Schreiben (E2330)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.86 Metadaten - Fehler: alte Version (E2334)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.87 Metadaten - Datei ist schreibgeschützt (E2331)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.88 Metadaten - Der Import-Gültigkeitsbereich ist mit dem Ausgabe-Gültigkeitsbereich nicht kompatibel (E2339)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.89 Metadaten - Kein wohlgeformter Name vergeben (E2332)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.90 Metadaten - Im Arbeitsspeicher oder Stream befinden sich keine .CLB-Daten (E2337)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.91 Metadaten - Datenbank ist schreibgeschützt (E2338)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.92 Metadaten - Fehler beim Öffnen des gemeinsam genutzten Speichers (E2335)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.93 Metadaten - Erzeugung des gemeinsam genutzten Speichers fehlgeschlagen Eine Speicherzuordnung mit demselben Namen ist bereits vorhanden. (E2336)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.94 Metadaten - Daten zu groß (E2344)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.95 Metadaten - Datenwert wurde abgeschnitten (E2333)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.96 Überkreuzender Bezug zweier Units auf '%s' (F2047)

Mehrere Units verweisen in ihren interface-Abschnitten aufeinander.

Während der Compiler den interface-Abschnitt einer Unit bearbeitet, bevor diese von einer anderen Unit verwendet werden kann, muss er die Compilierreihenfolge der interface-Abschnitte ermitteln können.

Prüfen Sie, ob wirkliche alle Units in den uses-Klauseln benötigt werden, und ob sie nicht stattdessen im implementation-Abschnitt eingebunden werden können.

```
unit A;
interface
uses B;           (*A verwendet B und B verwendet A*)
implementation
end.

unit B;
```

```
interface
uses A;
implementation
end.
```

Das Problem tritt auf, weil A und B sich in ihren interface-Abschnitten gegenseitig verwenden.

```
unit A;
interface
uses B;           (*Compilierreihenfolge: B.interface, A, B.implementation*)
implementation
end.
```

```
unit B;
interface
implementation
uses A;           (*In den implementation-Abschnitt verschoben*)
end.
```

Sie können diese Reihenfolge ändern, indem Sie die uses-Klausel in den implementation-Abschnitt verschieben.

4.1.3.6.97 Für den Zugriff auf Klasseneigenschaften muss ein Klassenfeld oder eine statische Klassenmethode verwendet werden (E2355)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.98 PROCEDURE, FUNCTION, PROPERTY oder VAR erwartet (E2123)

Nach "class" in einer Elementdeklaration innerhalb eines Klassentyps dürfen nur die Token procedure, function, var und property folgen.

4.1.3.6.99 Lokale Klassen- oder Interface-Typen sind nicht zulässig (E2061)

Entspricht object_local in früheren Compilern. Klassen- und Interface-Typen dürfen nicht innerhalb eines Prozedurenrumpfs deklariert werden.

4.1.3.6.100 Klassenelementdeklarationen nicht zulässig in anonymen Record- oder lokalen Record-Typ (E2435)

Record-Typen, die in lokalen Gültigkeitsbereichen oder in Variablendeklarationen deklariert sind, dürfen nur Felddeklarationen enthalten. Für erweiterte Features in Record-Typen (wie z.B. Methoden, Eigenschaften und verschachtelte Typen) muss der Record-Typ explizit als globaler Typ definiert werden.

4.1.3.6.101 Klassen- und Interface-Typen sind nur in Typabschnitten zulässig (E2060)

Klassen- oder Interface-Typen müssen immer mit einer expliziten Typdeklaration in einem Typabschnitt deklariert werden. Im Gegensatz zu Record-Typen dürfen sie nicht anonym sein.

Der Hauptgrund besteht darin, dass Sie sonst nicht die Methoden dieses Typs deklarieren könnten (weil kein Typname vorhanden ist).

Falsch (Versuch, einen Klassentyp in einer Variablendeklaration zu deklarieren):

```
program Produce;

var
  MyClass : class
    Field: Integer;
  end;

begin
end.
```

Richtig:

```
program Solve;

type
  TMyClass = class
    Field: Integer;
  end;

var
  MyClass : TMyClass;

begin
end.
```

4.1.3.6.102 %s-Klausel erwartet, aber %s gefunden (E2128)

Der Compiler erwartet aufgrund der Delphi-Syntax Klausel1, hat aber Klausel2 gefunden.

```
program Produce;

type
  CharDesc = class
    vch : Char;

  property Ch : Char;
    end;
  end.
```

Bei der ersten Deklaration einer Methode müssen Sie eine read- und eine write-Klausel angeben. Da bei der Eigenschaft Ch beide fehlen, tritt beim Compilieren ein Fehler auf. Ihre ursprüngliche Absicht lag vielleicht darin, einer in einer Basisklasse definierten Eigenschaft einen anderen Sichtbarkeitsstatus zu verleihen, z. B. private anstelle von public. In diesem Fall ist die wahrscheinlichste Fehlerursache, dass der Eigenschaftsname in der Basisklasse nicht gefunden wurde. Stellen Sie sicher, dass der Name der Eigenschaft richtig geschrieben wurde und die Eigenschaft in einer übergeordneten Klasse tatsächlich enthalten ist.

```
program Produce;

type
  CharDesc = class
    vch : Char;

  property Ch : Char read vch write vch;
    end;
  end.
```

Die Lösung besteht darin, alle benötigten Klauseln anzugeben.

4.1.3.6.103 Fehler beim Laden von .NET Framework %s: %08X (E2401)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.104 Importierter Bezeichner '%s' steht im Konflikt mit '%s' in '%s' (x2421)

Beim Importieren einer .NET-Assemblierung kann der Compiler auf Symbole stoßen, die nicht mit den CLS-Spezifikationen konform sind. Ein Beispiel dafür sind Bezeichner mit Unterscheidung der Groß-/Kleinschreibung im Gegensatz zu Bezeichnern ohne Unterscheidung der Groß-/Kleinschreibung. Ein weiteres Beispiel ist eine Eigenschaft in einer Klasse mit demselben Namen wie eine Methode oder ein Feld in derselben Klasse. Diese Fehlermeldung gibt an, dass gleichlautende Symbole im selben Gültigkeitsbereich (Elemente derselben Klasse oder desselben Interfaces) in einer importierten Assemblierung gefunden wurden und dass mit der Delphi-Syntax nur auf eines davon zugegriffen werden kann.

4.1.3.6.105 Importierter Bezeichner '%s' steht im Konflikt mit '%s' in Namespace '%s' (E2422)

Beim Importieren einer .NET-Assemblierung kann der Compiler auf Symbole stoßen, die nicht mit den CLS-Spezifikationen konform sind. Ein Beispiel dafür sind Bezeichner mit Unterscheidung der Groß-/Kleinschreibung im Gegensatz zu Bezeichnern ohne Unterscheidung der Groß-/Kleinschreibung. Ein weiteres Beispiel ist eine Eigenschaft in einer Klasse mit demselben Namen wie eine Methode oder ein Feld in derselben Klasse. Diese Fehlermeldung gibt an, dass gleichlautende Symbole im selben Gültigkeitsbereich (Elemente derselben Klasse oder desselben Interfaces) in einer importierten Assemblierung gefunden wurden und dass mit der Delphi-Syntax nur auf eines davon zugegriffen werden kann.

4.1.3.6.106 CLS: Überschreiben der virtuellen Methode '%s.%s'. Sichtbarkeit ('%s) muss mit der Basisklasse '%s' (%s) übereinstimmen (H2384)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.107 for-in-Anweisung arbeitet nicht mit Kollektionstyp '%s', weil '%s' kein Element für '%s' enthält oder darauf nicht zugegriffen werden kann (E2431)

Eine for-in-Anweisung kann nur mit den folgenden Kollektionstypen zusammenarbeiten:

- Primitive Typen, die der Compiler erkennt, wie z.B. Arrays, Mengen oder Strings
- Typen, die `IEnumerable` implementieren
- Typen, die das `GetEnumerator`-Pattern implementieren (siehe Delphi-Sprachreferenz)

Stellen Sie sicher, dass der angegebene Typ diesen Anforderungen entspricht.

Siehe auch

Deklarationen und Anweisungen (siehe Seite 862)

4.1.3.6.108 Vorzeichenbehaftete und -lose Typen werden kombiniert - beide Operanden werden erweitert (W1024)

Zum mathematischen Kombinieren von Typen mit und ohne Vorzeichen erweitert der Compiler zunächst beide Operanden auf einen nächstgrößeren gemeinsamen Datentyp.

Um zu sehen, warum die Typänderung erforderlich ist, betrachten Sie zwei Operanden, nämlich Integer mit dem Wert -128 und

Cardinal mit dem Wert 130. Der Typ Integer weist eine Stelle mehr an Genauigkeit auf als der Typ Cardinal, so dass der Vergleich der beiden Werte über nur 32 Bit nicht exakt erfolgen kann. Die Lösung für den Compiler besteht darin, beiden Typen auf eine nächsthöhere gemeinsame Größe zu erweitern und dann den Vergleich durchzuführen.

Der Compiler zeigt diese Warnung nur, wenn die Größe die zur Berechnung des Ergebnisses übliche Größe überschreitet.

```
{$APPTYPE CONSOLE}
program Produce;
var
  i : Integer;
  c : Cardinal;

begin
  i := -128;
  c := 130;
  WriteLn(i + c);
end.
```

In diesem Beispiel warnt der Compiler davor, dass der Ausdruck mit 64 Bit und nicht mit 32 Bit berechnet wird.

4.1.3.6.109 Vorzeichenbehaftete und -lose Typen werden verglichen - beide Operanden werden erweitert (W1023)

Zum korrekten Vergleichen von Typen mit und ohne Vorzeichen muss der Compiler beide Operanden auf den nächstgrößeren gemeinsamen Datentyp "hochstufen".

Um zu sehen, warum die Typänderung erforderlich ist, betrachten Sie zwei Operanden, nämlich Shortint mit dem Wert -128 und Byte mit dem Wert 130. Der Typ Byte weist eine Stelle mehr an Genauigkeit auf als der Typ Shortint, so dass der Vergleich der beiden Werte über nur 8 Bit nicht exakt erfolgen kann. Die Lösung für den Compiler besteht darin, beiden Typen auf eine nächsthöhere gemeinsame Größe zu erweitern und dann den Vergleich durchzuführen.

```
program Produce;
var
  s: shortint;
  b : byte;

begin
  s := -128;
  b := 130;

  assert(b < s);
end.
```

4.1.3.6.110 Der Vergleich ergibt immer False (W1021)

Der Compiler hat festgestellt, dass ein Ausdruck immer den Wert False liefert. Dies ist in den meisten Fällen das Ergebnis eines Randtests, wobei ein Vergleich mit einem bestimmten Variabtentyp erfolgt. Beispiel: Ein Integer verglichen mit \$80000000.

In früheren Versionen des Delphi-Compilers (vor 12.0) wäre die Hexadezimal-Konstante \$80000000 als negativer Integerwert definiert gewesen, aber durch die Einführung des Typs Int64 ist \$80000000 nun ein positiver Wert des Typs Int64. Dies hat zur Folge, dass Vergleiche dieser Konstante mit Integer-Variablen nicht mehr die gleichen Ergebnisse liefern wie in früheren Versionen.

Für den Umgang mit dieser Warnung gibt es kein Patentrezept. Manchmal kann sie ignoriert werden, aber in anderen Fällen ist es erforderlich, den Quelltext entsprechend zu ändern.

```
program Produce;
var
  i : Integer;
```

```

c : Cardinal;

begin
  c := 0;
  i := 0;
  if c < 0 then
    WriteLn('false');

  if i >= $80000000 then
    WriteLn('false');
end.

```

In diesem Beispiel ermittelt der Compiler, dass die beiden Ausdrücke immer den Wert False ergeben. Im ersten Fall ist ein Wert des Typs Cardinal, der kein Vorzeichen aufweist, immer größer oder gleich 0. Im zweiten Fall ist ein Wert des Typs 32-Bit-Integer immer kleiner als der Wert \$80000000 des Typs Int64.

4.1.3.6.111 Der Vergleich ergibt immer True (W1022)

Der Compiler hat festgestellt, dass ein Ausdruck immer den Wert True liefert. Dies ist in den meisten Fällen das Ergebnis eines Randtests, wobei ein Vergleich mit einem bestimmten Variablenwert erfolgt. Beispiel: Ein Integer verglichen mit \$80000000.

In früheren Versionen des CodeGear Pascal-Compilers (vor 12.0) wäre die Hexadezimal-Konstante \$80000000 als negativer Integerwert definiert gewesen, aber durch die Einführung des Typs Int64 ist \$80000000 nun ein positiver Wert des Typs Int64. Dies hat zur Folge, dass Vergleiche dieser Konstante mit Integer-Variablen nicht mehr die gleichen Ergebnisse liefern wie in früheren Versionen.

Für den Umgang mit dieser Warnung gibt es kein Patentrezept. Manchmal kann sie ignoriert werden, aber in anderen Fällen ist es erforderlich, den Quelltext entsprechend zu ändern.

```

program Produce;

var
  i : Integer;
  c : Cardinal;

begin
  c := 0;
  i := 0;
  if c >= 0 then
    WriteLn('true');

  if i < $80000000 then
    WriteLn('true');
end.

```

In diesem Beispiel ermittelt der Compiler, dass die beiden Ausdrücke immer den Wert True ergeben. Im ersten Fall ist ein Wert des Typs Cardinal, der kein Vorzeichen aufweist, immer größer oder gleich 0. Im zweiten Fall ist ein Wert des Typs 32-Bit-Integer immer kleiner als der Wert \$80000000 des Typs Int64.

4.1.3.6.112 Inkompatible Typen (E2008)

Diese Fehlermeldung wird angezeigt, wenn der Compiler zwei kompatible (sehr ähnliche) Typen erwartet, die Typen aber tatsächlich unterschiedlich sind. Dieser Fehler kann in sehr unterschiedlichen Situationen auftreten – beispielsweise, wenn eine Read- oder Write-Klausel in einer Eigenschaft eine Methode enthält, deren Parameterliste nicht mit der Eigenschaft übereinstimmt, oder wenn ein Parameter für eine Standardprozedur oder -methode den falschen Typ hat.

Dieser Fehler kann auch auftreten, wenn in zwei Units jeweils ein Typ mit demselben Namen deklariert wird. Diese Fehlermeldung kann zudem angezeigt werden, wenn eine Prozedur aus einer importierten Unit einen Parameter mit einem gleichnamigen Typ enthält, und eine Variable eines gleichnamigen Typs an die Prozedur übergeben wird.

```
4
unit unit1;
interface
  type
    ExportedType = (alpha, beta, gamma);

implementation
begin
end.

unit unit2;
interface
  type
    ExportedType = (alpha, beta, gamma);

  procedure ExportedProcedure(v : ExportedType);

implementation
  procedure ExportedProcedure(v : ExportedType);
  begin
  end;

begin
end.

program Produce;
uses unit1, unit2;

var
  A: array [0..9] of char;
  I: Integer;
  V : ExportedType;
begin
  ExportedProcedure(v);
  I:= Hi(A);
end.
```

Die Standardfunktion Hi erwartet ein Argument des Typs Integer oder Word, stattdessen wird ein Array verwendet. In dem Aufruf von ExportedProcedure ist V vom Typ unit1.ExportedType, da unit1 vorher in unit2 importiert wurde. Daher tritt ein Fehler auf.

```
unit unit1;
interface
  type
    ExportedType = (alpha, beta, gamma);

implementation
begin
end.

unit unit2;
interface
  type
    ExportedType = (alpha, beta, gamma);

  procedure ExportedProcedure(v : ExportedType);

implementation
  procedure ExportedProcedure(v : ExportedType);
  begin
  end;

begin
end.

program Solve;
uses unit1, unit2;
var
```

```

A: array [0..9] of char;
I: Integer;
V : unit2.ExportedType;
begin
  ExportedProcedure(v);
  I:= High(A);
end.

```

Eigentlich sollte die Standardfunktion High und nicht Hi verwendet werden. Für den Aufruf von ExportedProcedure gibt es zwei alternative Lösungen. Erstens könnten Sie die Reihenfolge der uses-Klausel ändern. Dies könnte aber zu ähnlichen Fehlern führen. Eine robustere Lösung ist, den Typenamen voll mit der Unit, die den gewünschten Typ deklariert, zu qualifizieren, wie es In diesem Beispiel für die Deklaration von V vorgenommen wurde.

4.1.3.6.113 Inkompative Typen: '%s' (E2009)

Der Compiler hat einen Unterschied zwischen der Deklaration und der Verwendung einer Prozedur entdeckt.

```

program Produce;

type
  ProcedureParm0 = procedure; stdcall;
  ProcedureParm1 = procedure(VAR x : Integer);

procedure WrongConvention; register;
begin
end;

procedure WrongParms(x, y, z : Integer);
begin
end;

procedure TakesParm0(p : ProcedureParm0);
begin
end;

procedure TakesParm1(p : ProcedureParm1);
begin
end;

begin
  TakesParm0(WrongConvention);
  TakesParm1(WrongParms);
end.

```

Der Aufruf von TakesParm0 löst einen Fehler aus, weil der Typ ProcedureParm0 eine Prozedur stdcall erwartet, während WrongConvention mit der Aufrufkonvention register deklariert wurde. Außerdem wird der Aufruf von TakesParm1 fehlschlagen, weil die Parameterlisten nicht übereinstimmen.

```

program Solve;

type
  ProcedureParm0 = procedure; stdcall;
  ProcedureParm1 = procedure(VAR x : Integer);

procedure RightConvention; stdcall;
begin
end;

procedure RightParms(VAR x : Integer);
begin
end;

procedure TakesParm0(p : ProcedureParm0);
begin

```

```

end;

procedure TakesParm1(p : ProcedureParm1);
begin
end;

begin
  TakesParm0(RightConvention);
  TakesParm1(RightParms);
end.

```

Die Lösung für beide Probleme liegt darin, darauf zu achten, dass die Aufrufkonvention bzw. die Parameterlisten mit der Deklaration übereinstimmen.

4

4.1.3.6.114 Inkompatible Typen: '%s' und '%s' (E2010)

Diese Fehlermeldung tritt auf, wenn der Compiler zwei Typen als kompatibel (oder sehr ähnlich) miteinander erwartet hat, sie sich aber als unterschiedlich herausgestellt haben.

```

program Produce;

procedure Proc(I: Integer);
begin
end;

begin
  Proc( 22 / 7 ); (*Ergebnis des /-Operators ist Real*)
end.

```

Hier ist ein C++ Programmierer davon ausgegangen, dass der Divisionsoperator "/" ein Ergebnis vom Typ Integer liefert – dies trifft in Delphi nicht zu.

```

program Solve;

procedure Proc(I: Integer);
begin
end;

begin
  Proc( 22 div 7 ); (*Der Operator div führt zu einem Ergebnis vom Typ Integer*)
end.

```

Die Lösung in diesem Fall liegt darin, den Operator für integrale Division div zu benutzen – im Allgemeinen müssen Sie Ihr Programm sehr gründlich betrachten, um entscheiden zu können, wie Typ-Inkompatibilitäten zu beheben sind.

4.1.3.6.115 Instanz von '%s' mit abstrakter Methode '%s.%s' wird konstruiert (x1020)

Der zu compilierende Quelltext konstruiert Instanzen von Klassen, die abstrakte Methoden enthalten.

```

program Produce;
(*$WARNINGS ON*)
(*$HINTS ON*)

type
  Base = class
    procedure Abstraction; virtual; abstract;
  end;

var
  b : Base;

```

```
begin
  b := Base.Create;
end.
```

Eine abstrakte Prozedur existiert nicht, so dass es gefährlich wird, Instanzen einer Klasse zu erstellen, die abstrakte Prozeduren enthält. In diesem Fall ist die Erstellung von b die Ursache für die Warnung. Jeder Aufruf von Abstraction während der hier erstellten Instanz von b würde zu einem Laufzeitfehler führen. Es wird ein Hinweis ausgegeben, dass der zu 'b' zugewiesene Wert niemals benutzt wird.

```
program Solve;
(*$WARNINGS ON*)
(*$HINTS ON*)

type
  Base = class
    procedure Abstraction; virtual;
  end;

var
  b : Base;

procedure Base.Abstraction;
begin
end;

begin
  b := Base.Create;
end.
```

Eine Lösung dieses Problems ist die Entfernung der abstrakten Anweisung aus der Deklaration der Prozedur, wie hier gezeigt wird. Eine andere Möglichkeit wäre, eine Klasse von Base abzuleiten und dann eine konkrete Version von Abstraction bereitzustellen. Es wird ein Hinweis ausgegeben, dass der zu 'b' zugewiesene Wert niemals benutzt wird.

4.1.3.6.116 Instanz der abstrakten Klasse '%s' wird konstruiert (E2402)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.117 Konstruktoren und Destruktoren sind im OLE-Automatisierungsbereich nicht zulässig (E2177)

Sie haben unzulässigerweise versucht, einen Konstruktor oder einen Destruktor in den OLE-Automatisierungsbereich einer Klassendeklaration zu stellen.

```
program Produce;

type
  Base = class
    automated
      constructor HardHatBob;
      destructor DemolitionBob;
    end;

    constructor Base.HardHatBob;
  begin
  end;

    destructor Base.DemolitionBob;
  begin
  end;

begin
```

end.

Es ist nicht möglich, einen Klassenkonstruktor oder -destruktor im OLE-Automatisierungsabschnitt zu deklarieren. Die Deklaration des Konstruktors und des Destruktors in diesem Quelltext lösen beide diesen Fehler aus.

```
program Solve;

type
  Base = class
    constructor HardHatBob;
    destructor DemolitionBob;
  end;

  constructor Base.HardHatBob;
begin
end;

  destructor Base.DemolitionBob;
begin
end;

begin
end.
```

Die einzige Lösung dieses Fehlers liegt darin, die Deklarationen aus dem Automatisierungsabschnitt herauszunehmen, wie dies in diesem Beispiel geschehen ist.

4.1.3.6.118 Konstantendeklarationen nicht zulässig in anonymen Record- oder lokalen Record-Typ (E2437)

Record-Typen, die in lokalen Gültigkeitsbereichen oder in Variablen Deklarationen deklariert sind, dürfen nur Felddeklarationen enthalten. Für erweiterte Features in Record-Typen (wie z.B. Methoden, Eigenschaften und verschachtelte Typen) muss der Record-Typ explizit als globaler Typ definiert werden.

4.1.3.6.119 Konstantenausdruck erwartet (E2026)

Der Compiler hat an dieser Stelle einen Konstantenausdruck erwartet, der gefundene Ausdruck war jedoch keine Konstante.

```
program Produce;
const
  Message = 'Hello World!';
  WPosition = Pos('W', Message);
begin
end.
```

Der Aufruf von Pos ist für den Compiler kein Konstantenausdruck, auch wenn die Argumente Konstanten sind und Pos im Prinzip während der Compilierung ausgewertet werden könnte.

```
program Solve;
const
  Message = 'Hello World!';
  WPosition = 7;
begin
end.
```

In diesem Fall müssen wir also einfach den rechten Wert für WPosition selbst berechnen.

4.1.3.6.120 Konstanten dürfen nicht als Argumente für offene Arrays verwendet werden (E2192)

Argumente für offene Array sind als Array-Variable, konstruiertes Array oder als einzelne Variable vom Elementtyp des Arguments zu übergeben..

```
program Produce;

procedure TakesArray(s : array of String);
begin
end;

begin TakesArray('Hello Error');
end.
```

Der Fehler in diesem Beispiel wird ausgelöst, weil ein Stringliteral anstelle des erwarteten Arrays angegeben wurde. Ein Array kann nicht implizit aus einer Konstanten konstruiert werden.

```
program Solve;

procedure TakesArray(s : array of String);
begin
end;

begin TakesArray(['Hello Error']);
end.
```

Mit der Lösung tritt kein Fehler auf, weil das Array explizit konstruiert wird.

4.1.3.6.121 Konstante oder Typenbezeichner erwartet (E2007)

Diese Fehlermeldung tritt auf, wenn der Compiler einen Typ erwartet, aber ein Symbol findet, dass weder eine Konstante ist (mit einer Konstante könnte ein Untermengentyp beginnen) noch ein Typbezeichner.

```
program Produce;
var
  c : ExceptionClass; (*ExceptionClass ist eine Variable in System*)
begin
end.
```

Hier ist ExceptionClass eine Variable, kein Typ.

```
program Solve;
program Produce;
var
  c : Exception; (*Exception ist ein Typ in SysUtils*)
begin
end.
```

Sie müssen sicherstellen, dass ein Typ festgelegt wird. Möglicherweise ist der Bezeichner falsch geschrieben oder wird von einem anderen Bezeichner, beispielsweise in einer anderen Unit, verborgen.

4.1.3.6.122 Konstantenobjekt kann nicht als Var-Parameter übergegeben werden (E2197)

Dies ist eine reservierte Fehlermeldung.

4.1.3.6.123 Es müssen C++-Obj.-Dateien generiert werden (-jp) (E2241)

Aufgrund der verwendeten Sprachmerkmale können keine Standard-C-Objektdateien für diese Unit erzeugt werden. Sie müssen C++ Objektdateien erzeugen.

4.1.3.6.124 CREATE erwartet (E2412)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.125 'Self' wurde mehr als einmal initialisiert (E2306)

Ein geerbter Konstruktor wurde mehrmals initialisiert.

4.1.3.6.126 'Self' ist nicht initialisiert. Ein geerbter Konstruktor muss aufgerufen werden (E2304)

In Delphi für .NET muss ein Konstruktor immer einen geerbten Konstruktor aufrufen, bevor er auf geerbte Klassenelemente zugreifen oder diese initialisieren kann (in Delphi für Win32 ist dies nicht der Fall). Fehlt der Aufruf des geerbten Konstruktors, führt dies zu einem Compiler-Fehler. Vor dem Aufruf des geerbten Konstruktors dürfen keinesfalls direkte oder indirekte Aufrufe von geerbten Klassenfeldern erfolgen.

Anmerkung: Ein Konstruktor kann jedoch Felder seiner eigenen Klasse initialisieren, bevor der geerbte Konstruktor aufgerufen wird.

Beispiel:

Für die Klasse

```
X=class
  constructor Create;
end;
```

ist ein geerbter Konstruktor in der Methode Create erforderlich:

```
constructor X.Create;
begin
  inherited Create;
end;
```

4.1.3.6.127 'Self' ist möglicherweise nicht initialisiert worden (E2305)

In Delphi für .NET muss ein Konstruktor immer einen geerbten Konstruktor aufrufen, bevor er auf geerbte Klassenelemente zugreifen oder diese initialisieren kann (in Delphi für Win32 ist dies nicht der Fall). Fehlt der Aufruf des geerbten Konstruktors, führt dies zu einem Compiler-Fehler. Vor dem Aufruf des geerbten Konstruktors dürfen keinesfalls direkte oder indirekte Aufrufe von geerbten Klassenfeldern erfolgen.

Anmerkung: Ein Konstruktor kann jedoch Felder seiner eigenen Klasse initialisieren, bevor der geerbte Konstruktor aufgerufen wird.

4.1.3.6.128 'Self' ist nicht initialisiert. Vor dem Zugriff auf das Vorfahrfeld '%s' muss ein geerbter Konstruktor aufgerufen werden (E2302)

In Delphi für .NET muss ein Konstruktor immer einen geerbten Konstruktor aufrufen, bevor er auf geerbte Klassenelemente zugreifen oder diese initialisieren kann (in Delphi für Win32 ist dies nicht der Fall). Fehlt der Aufruf des geerbten Konstruktors, führt dies zu einem Compiler-Fehler. Vor dem Aufruf des geerbten Konstruktors dürfen keinesfalls direkte oder indirekte Aufrufe von geerbten Klassenfeldern erfolgen.

Anmerkung: Ein Konstruktor kann jedoch Felder seiner eigenen Klasse initialisieren, bevor der geerbte Konstruktor aufgerufen wird.

4

4.1.3.6.129 'Self' ist nicht initialisiert. Vor dem Aufruf der Vorfahrmethode '%s' muss ein geerbter Konstruktor aufgerufen werden (E2303)

In Delphi für .NET muss ein Konstruktor immer einen geerbten Konstruktor aufrufen, bevor er auf geerbte Klassenelemente zugreifen oder diese initialisieren kann (in Delphi für Win32 ist dies nicht der Fall). Fehlt der Aufruf des geerbten Konstruktors, führt dies zu einem Compiler-Fehler. Vor dem Aufruf des geerbten Konstruktors dürfen keinesfalls direkte oder indirekte Aufrufe von geerbten Klassenfeldern erfolgen.

Anmerkung: Ein Konstruktor kann jedoch Felder seiner eigenen Klasse initialisieren, bevor der geerbte Konstruktor aufgerufen wird.

4.1.3.6.130 Bibliotheksname zu lang: %s (E2286)

Diese Meldung wird in diesem Produkt nicht verwendet.

4.1.3.6.131 Bei Umwandlung der angegebenen WideString-Konstante gehen Informationen verloren (H2455)

Alle Zeichen in einer WideString-Konstante, deren Ordinalwert größer als 127 ist, werden durch "?" ersetzt, wenn WideChar in der aktuellen, lokalen Codepage nicht dargestellt werden kann.

4.1.3.6.132 Durch Umwandlung der angegebenen WideChar-Konstante (#\$%04X) auf AnsiChar gehen Informationen verloren (H2451)

AnsiChar kann nur die ersten 256 Werte von WideChar darstellen, daher geht das zweite Byte von WideChar beim Konvertieren in AnsiChar verloren. Um diesen Informationsverlust zu vermeiden, sollten Sie WideChar Anteile von AnsiChar verwenden.

4.1.3.6.133 Für '%s' ist ein Vorgabewert erforderlich (E2238)

Bei Vorgabeparametern ist die Verwendung einer Parameterliste und eine anschließende Typangabe nicht erlaubt. Sie müssen jede Variable und die zugehörigen Standardwerte einzeln angeben.

```
program Produce;

procedure p0(a, b : Integer = 151);
begin
end;
```

```
begin
end.
```

Die Prozedurdefinition wird diesen Fehler hervorrufen, da sie zwei Parameter mit einem Standardwert deklariert.

```
program Solve;

procedure p0(a : Integer; b : Integer = 151);
begin
end;

procedure p1(a : Integer = 151; b : Integer = 151);
begin
end;

begin
end.
```

Es gibt zwei Möglichkeiten, dieses Problem zu lösen: Soll nur der letzte Parameter einen Standardwert erhalten, wenden Sie den Lösungsansatz im ersten Beispiel an. Sollen beide Parameter Standardwerte annehmen, verwenden Sie den zweiten Lösungsansatz.

4.1.3.6.134 Parameter '%s' ist hier wegen des Vorgabewerts nicht erlaubt (E2237)

Bei Vorgabeparametern ist die Verwendung einer Parameterliste und eine anschließende Typangabe nicht erlaubt. Sie müssen jede Variable und die zugehörigen Standardwerte einzeln angeben.

```
program Produce;

procedure p0(a, b : Integer = 151);
begin
end;

begin
end.
```

Die Prozedurdefinition wird diesen Fehler hervorrufen, da sie zwei Parameter mit einem Standardwert deklariert.

```
program Solve;

procedure p0(a : Integer; b : Integer = 151);
begin
end;

procedure p1(a : Integer = 151; b : Integer = 151);
begin
end;

begin
end.
```

Es gibt zwei Möglichkeiten, dieses Problem zu lösen: Soll nur der letzte Parameter einen Standardwert erhalten, wenden Sie den Lösungsansatz im ersten Beispiel an. Sollen beide Parameter Standardwerte annehmen, verwenden Sie den zweiten Lösungsansatz.

4.1.3.6.135 Standardeigenschaft muss eine Array-Eigenschaft sein (E2132)

Die Standardeigenschaft, die Sie für diese Klasse festlegen, ist keine Array-Eigenschaft. Standardeigenschaften müssen Array-Eigenschaften sein.

```

program Produce;

type
  Base = class
    function GetV : Char;
    procedure SetV(x : Char);

    property Data : Char read GetV write SetV; default;
  end;

  function Base.GetV : Char;
begin
  GetV := 'A';
end;

  procedure Base.SetV(x : Char);
begin
end;

begin
end.

```

Wenn Sie eine Standardeigenschaft festlegen, müssen Sie sicherstellen, dass sie der Syntax für Array-Eigenschaften entspricht. Die Eigenschaft Data im obigen Code hat jedoch den Typ Char.

```

program Solve;

type
  Base = class
    function GetV(i : Integer): Char;
    procedure SetV(i : Integer; const x : Char);

    property Data[i : Integer]: Char read GetV write SetV; default;
  end;

  function Base.GetV(i : Integer): Char;
begin
  GetV := 'A';
end;

  procedure Base.SetV(i : Integer; const x : Char);
begin
end;

begin
end.

```

Ändern Sie den Typ der Eigenschaft, oder entfernen Sie die Direktive default.

4.1.3.6.136 Der Vorgabeparameter '%s' muss als Wert oder Konstante übergeben werden (E2239)

Parameter, die Standardwerte sind, können nicht per Referenz übergeben werden.

```

program Produce;

  procedure p0(var x : Integer = 151);
begin
end;

begin
end.

```

Da der Parameter 'x' in diesem Beispiel per Referenz übergeben wird, darf er kein Standardwert sein.

```
program Solve;
```

```

procedure p0(const x : Integer = 151);
begin
end;

begin
end.

```

Eine Lösung besteht darin, den per Referenz übergebenen Parameter als Konstantenparameter zu übergeben. Alternativ dazu hätte man den Parameter über die Wertübergabe übergeben können oder der Standardwert hätte auch entfernt werden können.

4.1.3.6.137 Parameter dieses Typs dürfen keine Standardwerte haben (E2268)

Der in den Delphi-Compiler integrierte Mechanismus für Standardparameter ermöglicht nur die Initialisierung einfacher Typen auf diese Weise. Sie haben versucht, einen Typ zu benutzen, der nicht unterstützt wird.

```

program Produce;
type
  ArrayType = array [0..1] of integer;

  procedure p1(proc : ArrayType = [1, 2]);
begin
end;
end.

```

Standardparameter dieses Typ werden in Delphi nicht unterstützt.

```

program solve;
type
  ArrayType = array [0..1] of integer;

  procedure p1(proc : ArrayType);
begin
end;
end.

```

Die einzige Möglichkeit, diesen Fehler zu beseitigen, liegt darin, die ungültige Parameterzuweisung zu entfernen oder stattdessen einen Parametertyp festzulegen, der mit einem Standardwert initialisiert werden kann.

4.1.3.6.138 Klasse besitzt bereits eine Standardeigenschaft (E2131)

Sie versuchen, einer Klasse eine Standardeigenschaft zuzuweisen, die bereits über eine solche verfügt.

```

program Produce;

type
  Base = class
    function GetV(i : Integer): Char;
    procedure SetV(i : Integer; const x : Char);

    property Data[i : Integer]: Char read GetV write SetV; default;
    property Access[i : Integer]: Char read GetV write SetV; default;
  end;

  function Base.GetV(i : Integer): Char;
begin
  GetV := 'A';
end;

  procedure Base.SetV(i : Integer; const x : Char);
begin
end;

begin

```

```
end.
```

Die Eigenschaft Access im obigen Code soll zur Standardeigenschaft der Klasse werden. Dies ist aber bereits die Eigenschaft Data. In jeder Klasse ist nur eine Standardeigenschaft erlaubt.

```
program Solve;

type
  Base = class
    function GetV(i : Integer): Char;
    procedure SetV(i : Integer; const x : Char);

    property Data[i : Integer]: Char read GetV write SetV; default;
  end;

  function Base.GetV(i : Integer): Char;
begin
  GetV := 'A';
end;

procedure Base.SetV(i : Integer; const x : Char);
begin
end;

begin
end.
```

Entfernen Sie die ungültige Deklaration der Standardeigenschaft aus dem Code.

4.1.3.6.139 Standardwerte müssen vom Typ Ordinal, Pointer oder vom Typ small Set sein (E2146)

Sie haben eine Eigenschaft festgelegt, die eine default-Klausel enthält, aber der Typ der Eigenschaft kann nicht zusammen mit Standardwerten benutzt werden.

```
program Produce;

type
  VisualGauge = class
    pos : Single;
  property Position : Single read pos write pos default 0.0;
  end;

begin
end.
```

Das Programm erstellt eine Eigenschaft und versucht, ihr einen Standardwert zuzuweisen; da der Typ der Eigenschaft jedoch keine Standardwerte zulässt, wird eine Fehlermeldung ausgegeben.

```
program Produce;

type
  VisualGauge = class
    pos : Integer;
  property Position : Integer read pos write pos default 0;
  end;

begin
end.
```

Wenn dieser Fehler auftritt, gibt es zwei einfache Lösungen: Zum einen kann die Definition des Standardwertes entfernt werden, und zum anderen kann der Typ der Eigenschaft in einen Typ geändert werden, der einen Standardwert zulässt. Es kann allerdings vorkommen, dass Ihr Programm nicht so einfach in Ordnung zu bringen ist; bedenken Sie den Fall, wenn Sie eine zu große Mengen-Eigenschaft verwenden – in dieser Situation müssen Sie Ihr Programm sorgfältig untersuchen, um die beste

Lösung für das Problem feststellen zu können.

4.1.3.6.140 Unit System nicht kompatibel mit der Testversion (F2087)

Sie verwenden eine Testversion der Software. Diese ist zu der Anwendung, die Sie ausführen möchten, nicht kompatibel.

4.1.3.6.141 Destruktoren können nicht mit IDisposable gemischt werden (E2290)

Der Compiler erzeugt IDisposable-Unterstützung für eine Klasse, die einen Destruktor mit "Destroy" überschreibt. IDisposable kann nicht manuell implementiert und in derselben Klasse ein Destruktor implementiert werden.

4.1.3.6.142 Auf Ziel kann nicht zugegriffen werden (E2144)

Auf die Adresse, an die Sie einen Wert zu übergeben versuchten, kann aus der Entwicklungsumgebung heraus nicht zugegriffen werden.

4.1.3.6.143 Ziel kann nicht zugewiesen werden (E2453)

Der integrierte Debugger hat festgestellt, dass Ihre Zuweisung im aktuellen Kontext nicht gültig ist.

4.1.3.6.144 Unit '%s' wird mit Unit '%s' in '%s' kompiliert, aber abweichende Version von '%s' gefunden (F2446)

Dieser Fehler tritt auf, wenn eine Unit neu kompiliert werden muss, um Änderungen an einer anderen Unit zu übernehmen, aber die Quelle für die zu compilierende Unit nicht gefunden wird.

Anmerkung: Diese Fehlermeldung kann bei der Arbeit mit Inline-Funktionen auftreten. Die Erweiterung einer Inline-Funktion stellt deren Implementierung allen Units zur Verfügung, die diese Funktion aufrufen. Wenn eine Funktion *inline* ist, müssen Änderungen an dieser Funktion durch das Neucompilieren aller Units, die diese Funktion aufrufen, übernommen werden. Dies gilt auch dann, wenn alle Änderungen nur im **implementation**-Abschnitt vorkommen. Durch Inlining werden Ihre Units abhängiger voneinander und zur Aufrechterhaltung der binären Kompatibilität ist mehr Aufwand erforderlich. Das besonders für Entwickler wichtig, die .DCU-Dateien ohne den Quelltext weitergeben.

4.1.3.6.145 Die Direktive '%s' ist im Typ interface nicht erlaubt (E2210)

Beim Analysieren einer Schnittstelle wurde eine Direktive festgestellt. Direktiven dürfen aber nicht in Schnittstellen vorhanden sein.

```
program Produce;
  type
    IBaseIntf = interface
    private
      procedure fnord(x, y, z : Integer);
    end;

  begin
  end.
```

In diesem Beispiel gibt der Compiler eine Fehlermeldung aus, wenn er auf die Direktive `private` trifft, weil Direktiven in Schnittstellentypen nicht erlaubt sind.

```

program Solve;
type
  IBaseIntf = interface
    procedure fnord(x, y, z : Integer);
  end;

  TBaseClass = class (TInterfacedObject, IBaseIntf)
  private
    procedure fnord(x, y, z : Integer);
  end;

  procedure TBaseClass.fnord(x, y, z : Integer);
begin
end;
begin
end.

```

Das Problem kann nur auf eine Weise gelöst werden: Sie müssen die betreffende Direktive aus der Schnittstellendefinition entfernen. Schnittstellen unterstützen diese Direktiven nicht. Sie können aber die implementierende Methode in den gewünschten Sichtbarkeitsabschnitt einfügen. In diesem Beispiel wird das Ziel erreicht, indem die Prozedur TBaseClass.fnord in einen private-Abschnitt eingefügt wird.

4.1.3.6.146 **dispid-Klausel nur im OLE-Automatisierungsbereich erlaubt (E2183)**

Ein Dispid wurde einer Eigenschaft gegeben, die sich nicht in einem Abschnitt automated befindet.

```

program Produce;

type
  Base = class
    v : integer;
    procedure setV(x : integer);
    function getV : integer;
    property Value : integer read getV write setV dispid 151;
  end;

  procedure Base.setV(x : integer);
begin v := x;
end;

  function Base.getV : integer;
begin getV := v;
end;

begin
end.

```

Dieses Programm versucht, dispid für ein OLE-Automatisierungsobjekt zu setzen, aber die Eigenschaft wurde nicht in einem Automatisierungsabschnitt deklariert.

```

program Solve;

type
  Base = class
    v : integer;
    procedure setV(x : integer);
    function getV : integer;
  automated
    property Value : integer read getV write setV dispid 151;
  end;

  procedure Base.setV(x : integer);

```

```

begin v := x;
end;

function Base.getV : integer;
begin getV := v;
end;

begin
end.

```

Um diesen Fehler zu beheben, können Sie entweder die dispid-Anweisung aus der Deklaration der Eigenschaft entfernen oder die Deklaration der Eigenschaft in einen Automatisierungsabschnitt verlegen.

4.1.3.6.147 Das Eigenschaftsattribut 'label' kann in dispinterface nicht verwendet werden (E2274)

Sie haben einer in einer dispinterface-Schnittstelle definierten Eigenschaft ein Label hinzugefügt. Dies ist aber entsprechend der Sprachdefinition unzulässig.

```

program Problem;

type
  T0 = dispinterface
    ['{15101510-1510-1510-1510-151015101510}']
    function R : Integer;
    property value : Integer label 'Key';
  end;

begin
end.

```

Hier wird versucht, ein Label-Attribut in einer dispinterface-Eigenschaft zu verwenden.

```

program Solve;

type
  T0 = dispinterface
    ['{15101510-1510-1510-1510-151015101510}']
    function R : Integer;
    property value : Integer;
  end;

begin
end.

```

Das Problem kann nur behoben werden, wenn das Label-Attribut aus der Eigenschaftsdefinition entfernt wird.

4.1.3.6.148 Disposed_ darf nicht in Klassen mit Destruktoren deklariert werden (E2414)

Disposed_ darf nicht in Klassen mit Destruktoren deklariert werden. Wenn eine Klasse ein IDisposable-Interface implementiert, erzeugt der Compiler ein Feld namens Disposed_, um festzustellen, ob die Methode IDisposable.Dispose bereits aufgerufen wurde.

4.1.3.6.149 Prozedur DISPOSE benötigt einen Destruktor (E2080)

Diese Fehlermeldung wird angezeigt, wenn Sie in der Parameterliste einen Bezeichner an Dispose übergeben, der kein Destruktor ist.

```
program Produce;
```

```

type
  PMyObject = ^TMyObject;
  TMyObject = object
    F: Integer;
    constructor Init;
    destructor Done;
  end;

constructor TMyObject.Init;
begin
  F := 42;
end;

destructor TMyObject.Done;
begin
end;

var
  P: PMyObject;

begin
  New(P, Init);
  (*...*)
  Dispose(P, Init);           (*<-- Hier die Fehlermeldung*)
end.

```

Dispose wurde versehentlich mit dem Konstruktor anstelle des Destruktors aufgerufen.

program Solve;

```

type
  PMyObject = ^TMyObject;
  TMyObject = object
    F: Integer;
    constructor Init;
    destructor Done;
  end;

constructor TMyObject.Init;
begin
  F := 42;
end;

destructor TMyObject.Done;
begin
end;

var
  P: PMyObject;

begin
  New(P, Init);
  Dispose(P, Done);
end.

```

Übergeben Sie entweder den Destruktor oder kein zweites Argument.

4.1.3.6.150 Ein Typ dispinterface kann keinen Vorfahr interface haben (E2228)

Ein mit dispinterface spezifizierter Schnittstellentyp darf keine Vorfahrschnittstelle besitzen.

program Produce;

```

type
  IBase = interface

```

```

end;

IExtend = dispinterface (IBase)
[ '{00000000-0000-0000-0000-000000000000}' ]

end;

begin
end.

In diesem Beispiel führt der Versuch, für IExtend eine Vorfahrtschnittstelle zu spezifizieren, zu einem Compilierungsfehler.

program Solve;

type
  IBase = interface
  end;

  IExtend = dispinterface
  [ '{00000000-0000-0000-0000-000000000000}' ]

  end;

begin
end.

```

Es gibt zwei Möglichkeiten, diesen Fehler zu beseitigen: Entweder Sie entfernen die Deklaration für die Vorfahrtschnittstelle, oder Sie wandeln die dispinterface-Schnittstelle in eine reguläre Schnittstelle um. Im vorliegenden Beispiel wird die erste Vorgehensweise verwendet.

4.1.3.6.151 Methoden des Typs dispinterface dürfen keine Direktiven spezifizieren (E2230)

Methoden, die in einem dispinterface-Typ deklariert sind, können keine Direktiven für die Aufrufkonvention festlegen.

```

program Produce;

type
  IBase = dispinterface
  [ '{00000000-0000-0000-0000-000000000000}' ]
    procedure yamadama; register;
  end;

begin
end.

```

Im vorliegenden Beispiel führt der Versuch, in der Methode yamadama die Aufrufkonvention register festzulegen, zu einem Fehler.

```

program Solve;

type
  IBase = dispinterface
  [ '{00000000-0000-0000-0000-000000000000}' ]
    procedure yamadama;
  end;

begin
end.

```

Da eine dispinterface-Methode keine Direktiven für Aufrufkonventionen enthalten darf, kann das Problem nur durch Entfernen der Fehler verursachenden Direktive beseitigt werden (siehe Beispiel).

4.1.3.6.152 Ein Typ dispinterface benötigt eine interface-Identifikation (E2229)

Beim Einsatz eines dispinterface-Typs muss sichergestellt sein, dass eine entsprechende GUID-Spezifikation existiert.

```
program Produce;

type
  IBase = dispinterface
  end;

begin
end.
```

In diesem Beispiel ist für den dispinterface-Typ keine GUID-Angabe vorhanden. Dies führt zur Ausgabe einer Fehlermeldung durch den Compiler.

```
program Solve;

type
  IBase = dispinterface
  ['{00000000-0000-0000-0000-000000000000}']
  end;

begin
end.
```

Durch die Zuweisung einer GUID an die dispinterface-Schnittstelle lässt sich das Problem beseitigen.

4.1.3.6.153 Division durch Null (E2098)

Der Compiler hat in Ihrem Programm bei einem konstanten Ausdruck eine Division durch Null entdeckt.

Überprüfen Sie Ihre konstanten Ausdrücke.

4.1.3.6.154 Ein DLLImport-Attribut und eine externe oder Aufrufkonventionsdirektive dürfen nicht zusammen verwendet werden (E2293)

Der Compiler gibt intern DLLImport-Attribute für externe Funktionsdeklarationen aus. Dieser Fehler wird ausgelöst, wenn Sie ein eigenes DLLImport-Attribut für eine Funktion deklarieren und die externe Namensklausel für die Funktion verwenden.

4.1.3.6.155 Doppelte Symbolnamen im Namespace. '%s.%s' in %s gefunden. Duplikat in %s wird ignoriert (W1051)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.156 Doppeltes Case-Label (E2030)

Diese Fehlermeldung tritt auf, wenn mehr als ein Case-Label mit dem angegebenen Wert in einer Case-Anweisung existiert.

```
program Produce;

function DigitCount(I: Integer): Integer;
begin
  case Abs(I) of
    0:           DigitCount := 1;
```

```

0      ..9:      DigitCount := 1;  (*<-- Hier die Fehlermeldung*)
10     ..99:      DigitCount := 2;
100    ..999:      DigitCount := 3;
1000   ..9999:      DigitCount := 4;
10000  ..99999:      DigitCount := 5;
100000 ..999999:      DigitCount := 6;
1000000 ..9999999:      DigitCount := 7;
10000000 ..99999999:      DigitCount := 8;
100000000 ..999999999:      DigitCount := 9;
else
  DigitCount := 10;
end;
begin
  Writeln( DigitCount(12345) );
end.

```

Hier haben wir nicht aufgepasst und den Case-Label 0 zweimal genannt.

```

program Solve;

function DigitCount(I: Integer): Integer;
begin
  case Abs(I) of
    0      ..9:      DigitCount := 1;
    10     ..99:      DigitCount := 2;
    100    ..999:      DigitCount := 3;
    1000   ..9999:      DigitCount := 4;
    10000  ..99999:      DigitCount := 5;
    100000 ..999999:      DigitCount := 6;
    1000000 ..9999999:      DigitCount := 7;
    10000000 ..99999999:      DigitCount := 8;
    100000000 ..999999999:      DigitCount := 9;
  else
    DigitCount := 10;
  end;
begin
  Writeln( DigitCount(12345) );
end.

```

Im Allgemeinen ist das Problem nicht so einfach zu finden, wenn Sie Symbolkonstanten und Bereiche von Case-Labels verwenden – gegebenenfalls müssen Sie die tatsächlichen Werte der Konstanten notieren, um die Fehlerursache feststellen zu können.

4.1.3.6.157 %s '%s' doppelt mit identischen Parametern; Zugriff von C++ nicht möglich (W1029)

Es wird eine Objektdatei erzeugt, und zwei unterschiedlich benannte Konstruktoren oder Destruktoren mit identischen Parameterlisten wurden erzeugt. Auf sie kann bei der Umsetzung des Quelltext in eine HPP-Datei nicht zugegriffen werden, weil sowohl die Konstruktor- als auch die Destruktornamen in den Klassennamen konvertiert werden. In C++ erscheinen diese doppelt vorhandenen Deklarationen als ein und dieselbe Funktion.

```

unit Produce;
interface
  type
    Base = class
      constructor ctor0(a, b, c : integer);
      constructor ctor1(a, b, c : integer);
    end;

  implementation
  constructor Base.ctor0(a, b, c : integer);

```

```

begin
end;

constructor Base.ctor1(a, b, c : integer);
begin
end;

begin
end.

```

In diesem Beispiel weisen die beiden Konstruktoren die gleiche Signatur auf. Beim Compilieren der Datei mit einer der -j-Optionen wird eine Warnung generiert.

```

unit Solve;
interface
type
  Base = class
    constructor ctor0(a, b, c : integer);
    constructor ctor1(a, b, c : integer; dummy : integer = 0);
  end;

implementation
constructor Base.ctor0(a, b, c : integer);
begin
end;

constructor Base.ctor1(a, b, c : integer; dummy : integer);
begin
end;

begin
end.

```

Eine einfache Lösung besteht darin, die Signatur eines der Konstruktoren zu ändern, beispielsweise durch Hinzufügen eines zusätzlichen Parameters. Im Beispiel wurde ein Standardparameter zu ctor1 hinzugefügt. Diese Lösung hat den Vorteil, dass vorhandener Delphi-Quelltext, der Ctor1 verwendet, nicht geändert zu werden muss. Andererseits muss im C++ Quelltext ein zusätzlicher Parameter angegeben werden, der dem Compiler mitteilt, welcher Konstruktor verwendet werden soll.

4.1.3.6.158 dispid '%d' wird bereits von '%s' verwendet (E2180)

Es wurde versucht, dispid zu benutzen, das bereits einem anderen Mitglied dieser Klasse zugewiesen wurde.

```

program Produce;

type
  Base = class
    v : Integer;
    procedure setV(x : Integer);
    function getV : Integer;
  end;

  automated
    property Value : Integer read getV write setV dispid 151;
    property SecondValue : Integer read getV write setV dispid 151;
  end;

procedure Base.setV(x : Integer);
begin v := x;
end;

function Base.getV : Integer;
begin getV := v;
end;

begin
end.

```

Alle Dispide einer automatisierten Eigenschaft müssen eindeutig sein; SecondValue ist daher fehlerhaft.

```
program Solve;

type
  Base = class
    v : Integer;
    procedure setV(x : Integer);
    function getV : Integer;
  automated
    property Value : Integer read getV write setV dispid 151;
    property SecondValue : Integer read getV write setV dispid 152;
  end;

  procedure Base.setV(x : Integer);
begin v := x;
end;

  function Base.getV : Integer;
begin getV := v;
end;

begin
end.
```

Der Fehler wird mit einem eindeutigen dispid für SecondValue behoben.

4.1.3.6.159 Methode '%s' mit identischen Parametern und Ergebnistyp ist bereits vorhanden (E2301)

Innerhalb einer Klasse dürfen nicht mehrere überladene Methoden mit demselben Namen als **published** deklariert werden. Die Pflege von Typinformationen zur Laufzeit bedarf für jedes als **published** deklariertes Element eines eindeutigen Namens.

```
type
  TSomeClass = class
  published
    function Func(P: Integer): Integer;
    function Func(P: Boolean): Integer; // Fehler
```

4.1.3.6.160 Doppelte implements-Klausel für Interface '%s' (E2257)

Der Compiler hat zwei unterschiedliche Eigenschaftendeklarationen gefunden, die beide angeben, dieselbe Schnittstelle zu implementieren. Eine Schnittstelle kann jedoch nur durch eine Eigenschaft implementiert werden.

```
program Produce;
type
  IMyInterface = interface
  end;

  TMyClass = class(TInterfacedObject, IMyInterface)
    FMyInterface: IMyInterface;
    property MyInterface: IMyInterface read FMyInterface implements IMyInterface;
    property OtherInterface: IMyInterface read FMyInterface implements IMyInterface;
  end;
end.
```

In diesem Beispiel versucht jede der beiden Deklarationen MyInterface und OtherInterface die Schnittstelle IMyInterface zu implementieren. Die betreffende Schnittstelle kann jedoch nur von einer Eigenschaft implementiert werden.

Die einzige Lösung besteht darin, eine der beiden implements-Klauseln zu entfernen.

4.1.3.6.161 Doppeltes Symbol '%s' im Namespace '%s' von '%s' und '%s' definiert (E2447)

Dieser Fehler tritt auf, wenn Symbole aus verschiedenen Units in einem gemeinsamen Namespace kombiniert werden und derselbe Symbolname mehrmals vorhanden ist. In früheren Versionen von Delphi konnten diese Units ohne Fehler compiliert werden, weil der Gültigkeitsbereich der Symbole nur durch die Unit definiert wurde. In RAD Studio müssen Units zum Erzeugen von IL-Metadaten in Namespaces eingefügt werden. Dadurch könnten mehrere Units in einem einzigen Namespace kombiniert werden.

Sie beheben dieses Problem, indem Sie Symbole umbenennen, ein Symbol als Alias einem anderen zuweisen oder die Unitnamen ändern, so dass sie nicht in demselben Namespace kombiniert werden.

4.1.3.6.162 Doppelter Botschaftsmethoden-Index (E2140)

Sie haben für eine dynamische Methode einen Index angegeben, der bereits von einer anderen dynamischen Methode verwendet wird.

```
program Produce;

type
  Base = class
    procedure First(VAR x : Integer); message 151;
    procedure Second(VAR x : Integer); message 151;
  end;

  procedure Base.First(VAR x : Integer);
begin
end;

  procedure Base.Second(VAR x : Integer);
begin
end;

begin
end.
```

Die Deklaration von Second versucht, denselben Botschaftsindex zu verwenden, der bereits von First verwendet wird. Das ist nicht erlaubt.

```
program Solve;

type
  Base = class
    procedure First(VAR x : Integer); message 151;
    procedure Second(VAR x : Integer); message 152; (*eindeutiger Index*)
  end;

  Derived = class (Base)
    procedure First(VAR x : Integer); override; (*Verhalten der Basisklasse überschreiben*)
  end;

  procedure Base.First(VAR x : Integer);
begin
end;

  procedure Base.Second(VAR x : Integer);
begin
end;

  procedure Derived.First(VAR x : Integer);
```

```
begin
end;
```

```
begin
end.
```

Für dieses Problem gibt es zwei Lösungen. Wenn Sie denselben Botschaftswert tatsächlich verwenden müssen, können Sie die Botschaftsnummer so ändern, dass sie eindeutig ist. Sie können eine neue Klasse von der Basisklasse ableiten und das Verhalten der dort deklarierten Botschaftsbehandlung ändern. Das obige Beispiel zeigt beide Möglichkeiten.

4.1.3.6.163 Es gibt bereits eine Methode '%s' mit identischen Parametern (E2252)

Im betreffenden Datentyp gibt es bereits eine Methode mit der gleichen Signatur.

```
program Produce;

type
  t0 = class
    procedure f0(a : integer); overload;
    procedure f0(a : integer); overload;
  end;

procedure T0.f0(a : integer);
begin
end;

begin
end.
```

Hier entsteht der Fehler, weil es zwei überladene Deklarationen für dieselbe Prozedur gibt.

```
program Solve;

type
  t0 = class
    procedure f0(a : integer); overload;
    procedure f0(a : char); overload;
  end;

procedure T0.f0(a : integer);
begin
end;

procedure T0.f0(a : char);
begin
end;

begin
end.
```

Zur Behebung dieses Fehlers gibt es mehrere Möglichkeiten: Einerseits können Sie die redundante Deklaration aus der Prozedur entfernen. Ein anderer Ansatz, auf den hier zurückgegriffen wird, besteht darin, den Parametertyp der mehrfach vorhandenen Deklarationen so zu ändern, dass eine eindeutige Version der überladenen Prozedur erzeugt wird.

4.1.3.6.164 Nur eine Methode aus einer Gruppe überladener Methoden darf 'published' sein (E2266)

Nur ein Element aus einer Gruppe überladener Funktionen kann als published deklariert werden, weil die für Prozeduren generierten Laufzeit-Typinformationen (RTTI) nur den Namen enthalten.

```
(*$M+*)
(*$APPTYPE CONSOLE*)
program Produce;
type
  Base = class
  published
    procedure p1(a : integer); overload;
    procedure p1(a : boolean); overload;
  end;

  Extended = class (Base)
    procedure e1(a : integer); overload;
    procedure e1(a : boolean); overload;
  end;

procedure Base.p1(a : integer);
begin
end;

procedure Base.p1(a : boolean);
begin
end;

procedure Extended.e1(a : integer);
begin
end;

procedure Extended.e1(a : boolean);
begin
end;

end.
```

Im hier gezeigten Beispiel sind die beiden überladenen p1-Funktionen in einem als published deklarierten Abschnitt enthalten, was nicht zulässig ist.

Da außerdem der Status \$M+ verwendet wird, beginnt die Klasse Extended mit einer als published deklarierten Sichtbarkeit, so dass der Fehler auch bei dieser Klasse auftritt.

```
(*$M+*)
(*$APPTYPE CONSOLE*)
program Solve;
type
  Base = class
  public
    procedure p1(a : integer); overload;
  published
    procedure p1(a : boolean); overload;
  end;

  Extended = class (Base)
  public
    procedure e1(a : integer); overload;
    procedure e1(a : boolean); overload;
  end;

procedure Base.p1(a : integer);
begin
end;

procedure Base.p1(a : boolean);
begin
end;

procedure Extended.e1(a : integer);
begin
end;
```

```

end;

procedure Extended.e1(a : boolean);
begin
end;

end.

```

Die Lösung in diesem Fall besteht darin, sicherzustellen, dass nicht mehr als ein Element einer Gruppe von überladenen Funktionen in einem als published deklarierten Abschnitt enthalten ist. Am einfachsten erreichen Sie dies, indem Sie die Sichtbarkeit als public, protected oder private deklarieren.

4.1.3.6.165 Doppelte Ressourcen-ID: Typ %d ID %d (E2285)

Eine zum Projekt gelinkte Ressource stimmt in ihrem Typ und Namen oder in ihrem Typ und ihrer Ressourcen-ID mit einer anderen, zum Projekt gelinkten Ressource überein. (Delphi gibt eine Warnung aus und ignoriert den Fehler. in Kylix verursachen nicht eindeutige Ressourcen einen Fehler.)

4.1.3.6.166 Doppelten Ressourcenbezeichner %s in Unit %s(%s) und %s(%s) gefunden (E2407)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.167 Doppelter Ressourcenname: Typ %d '%s' (E2284)

Eine zum Projekt gelinkte Ressource stimmt in ihrem Typ und Namen oder in ihrem Typ und ihrer Ressourcen-ID mit einer anderen, zum Projekt gelinkten Ressource überein. (Delphi gibt eine Warnung aus und ignoriert den Fehler. in Kylix verursachen nicht eindeutige Ressourcen einen Fehler.)

4.1.3.6.168 Doppelte Implementierung für 'set of %s' in diesem Gültigkeitsbereich (E2429)

Um diesen Fehler zu vermeiden, deklarieren Sie einen expliziten Set-Typbezeichner anstelle von anonymen In-Place-Set-Ausdrücken.

4.1.3.6.169 Doppelter Tag-Wert (E2027)

Diese Fehlermeldung wird angezeigt, wenn eine Konstante mehr als einmal in der Deklaration eines Variant-Datensatzes erscheint.

```

program Produce;
type
  VariantRecord = record
    case Integer of
      0: (IntegerField: Integer);
      0: (RealField: Real);      (*--- Hier die Fehlermeldung*)
  end;

begin
end.
  program Solve;
type
  VariantRecord = record
    case Integer of

```

```
0: (IntField: Integer);
1: (RealField: Real);
end;

begin
end.
```

4.1.3.6.170 Namespace-Konflikte mit Unit-Name '%s' (E2399)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.171 Dynamischer Array-Typ erforderlich (E2413)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.172 Dynamische und Botschaftsmethoden sind im OLE-Automatisierungsbereich nicht erlaubt (E2178)

Sie haben unzulässigerweise eine dynamische Methode oder eine Botschaftsmethode in einen Automatisierungsbereich einer Klassendeklaration gestellt.

```
program Produce;

type
  Base = class
  automated
    procedure DynaMethod; dynamic;
    procedure MessageMethod(VAR msg : Integer); message 151;
  end;

  procedure Base.DynaMethod;
begin
end;

  procedure Base.MessageMethod;
begin
end;

begin
end.
```

Es ist nicht möglich, eine Deklaration einer dynamischen Methode oder einer Botschaftsmethode in den OLE-Automatisierungsabschnitt einer Klasse zu stellen. Aus diesem Grunde erzeugen beide Methodendeklarationen in diesem Quelltext einen Fehler.

```
program Solve;

type
  Base = class
    procedure DynaMethod; dynamic;
    procedure MessageMethod(VAR msg : Integer); message 151;
  end;

  procedure Base.DynaMethod;
begin
end;

  procedure Base.MessageMethod;
begin
end;
```

```
begin
end.
```

Dieser Fehler kann auf verschiedene Arten aus Ihrem Programm entfernt werden. Als Erstes könnten Sie alle Deklarationen, die diesen Fehler auslösen, aus dem Automatisierungsabschnitt herausbringen, wie dies in diesem Beispiel geschehen ist. Eine andere Möglichkeit besteht darin, die Attribut `dynamic` oder `message` der Methode zu entfernen; dadurch wird natürlich nicht das gewünschte Verhalten der Methode erzielt, aber der Fehler ist behoben.

4.1.3.6.173 Fehler beim Konvertieren der Ressource %s (E2378)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4

4.1.3.6.174 Fehler beim Signieren der Assemblierung (E2385)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.175 EXCEPT oder FINALLY erwartet (E2125)

Der Compiler erwartet im Exception-Behandlungscode eines der Schlüsselwörter `finally` oder `except`.

```
program Produce;

begin
  try
  end;
end.
```

In diesem Code fehlt die `except`- oder `finally`-Klausel des Exception-Behandlungscodes. Daher gibt der Compiler einen Fehler aus.

```
program Solve;

begin
  try
  except
  end;
end.
```

Fügen Sie die fehlende Klausel hinzu, damit der Code compiliert werden kann. Im obigen Beispiel wurde das Problem durch die eingefügte `except`-Klausel behoben.

4.1.3.6.176 %s erwartet, aber %s gefunden (E2029)

Diese Fehlermeldung erscheint bei Syntaxfehlern. Wahrscheinlich ist in der Quelle ein Schreibfehler aufgetreten, oder es wurde etwas ausgelassen. Wenn der Fehler am Beginn einer Zeile erscheint, befindet sich der eigentliche Fehler häufig in der vorangegangenen Zeile.

```
program Produce;
var
  I: Integer
begin           (*<-- Hier die Fehlermeldung ';' erwartet, aber 'BEGIN' gefunden*)
end.
```

Nach dem Typ `Integer` erwartet der Compiler ein Semikolon, mit dem die Deklaration der Variablen abgeschlossen wird. Er findet das Semikolon nicht in der aktuellen Zeile, liest also weiter und findet das Schlüsselwort `begin` am Anfang der nächsten Zeile. Zu diesem Zeitpunkt weiß der Compiler endgültig, dass etwas nicht stimmt ...

```
program Solve;
```

```

var
  I: Integer;      (*Semikolon hat gefehlt*)
begin
end.

```

In diesem Fall fehlte lediglich das Semikolon – ein häufiger Fehler in der Praxis. Im Allgemeinen sollten Sie die Zeile genau untersuchen, in der der Fehler aufgetreten ist, und dazu die Zeile darüber, um festzustellen, ob etwas fehlt oder falsch geschrieben wurde.

4.1.3.6.177 EXPORTS nur im globalen Bereich erlaubt (E2191)

Im Programm wurde eine Anweisung exports mit nichtglobalem Gültigkeitsbereich gefunden.

```

program Produce;

procedure ExportedProcedure;
exports ExportedProcedure;
begin
end;

begin
end.

```

Eine exports-Anweisung ist ausschließlich mit globalem Gültigkeitsbereich zulässig.

```

program Solve;

procedure ExportedProcedure;
begin
end;

exports ExportedProcedure;
begin
end.

```

Die Lösung liegt darin, darauf zu achten, dass Ihre EXPORTS-Anweisung globalen Gültigkeitsbereich hat und textlich allen in der Anweisung aufgeführten Prozeduren folgt. Allgemein gilt, dass EXPORTS-Anweisungen am besten unmittelbar vor den Quelltext zur Initialisierung der Quelldatei gestellt werden.

4.1.3.6.178 Ausdruck hat keinen Wert (E2143)

Sie haben versucht, das Ergebnis eines Ausdrucks, der keinen Wert erzeugt, einer Variablen zuzuweisen.

4.1.3.6.179 Sealed Klasse '%s' kann nicht erweitert werden (E2353)

Mit dem Modifizierer sealed wird die Vererbung (und somit die Erweiterung) einer Klasse verhindert.

4.1.3.6.180 Prozedur FAIL nur im Konstruktor erlaubt (E2078)

Die Standardprozedur Fail darf nur in einem Konstruktor aufgerufen werden.

4.1.3.6.181 Felddefinition nicht erlaubt nach Methoden oder Eigenschaften (E2169)

Sie haben versucht, einer Klasse weitere Felder hinzuzufügen, nachdem bereits eine Methode oder Prozedur deklariert wurde. Alle Felddefinitionen müssen vor die Methoden und Eigenschaften gestellt werden.

```

program Produce;

type
  Base = class
    procedure FirstMethod;
    a: Integer;
  end;

procedure Base.FirstMethod;
begin
end;

begin
end.

```

Die Deklaration von a nach FirstMethod löst einen Fehler aus.

```

program Solve;

type
  Base = class
    a: Integer;
    procedure FirstMethod;
  end;

procedure Base.FirstMethod;
begin
end;

begin
end.

```

Um diesen Fehler zu beheben, reicht es in der Regel aus, alle Felddefinitionen vor die erste Deklaration einer Methode oder einer Eigenschaft zu verschieben.

4.1.3.6.182 Felddefinition nicht erlaubt in OLE-Automatisierungsbereich (E2175)

Sie haben versucht, eine Felddefinition in einen OLE-Automatisierungsabschnitt einer Klassendeklaration zu platzieren. Im Abschnitt automated dürfen nur Eigenschaften und Methoden deklariert werden.

```

program Produce;

type
  Base = class
  automated
    i : Integer;
  end;

begin
end.

```

Die Deklaration von 'i' in dieser Klasse löst den Compilierungsfehler aus.

```

program Solve;

type
  Base = class
    i : Integer;
  automated
  end;

begin

```

```
end.
```

Durch Verschiebung der Deklaration von 'i' aus dem Automatisierungsabschnitt heraus wird der Fehler behoben.

4.1.3.6.183 Instanzelement '%s' in diesem Zusammenhang nicht verfügbar (E2124)

Sie versuchen, ein Instanzelement aus einer Klassenprozedur heraus zu referenzieren.

```
program Produce;

type
  Base = class
    Title : String;

    class procedure Init;
  end;

  class procedure Base.Init;
begin
  Self.Title := 'Does not work';
  Title := 'Does not work';
end;

begin
end.
```

Da Klassenprozeduren keine Instanzzeiger haben, können Sie nicht auf Methoden oder Instanzdaten der Klasse zugreifen.

```
program Solve;

type
  Base = class
    Title : String;

    class procedure Init;
  end;

  class procedure Base.Init;
begin
end;

begin
end.
```

Die einzige Lösung für dieses Problem besteht darin, aus einer Klassenmethode heraus nicht auf Elementdaten oder Elementmethoden zuzugreifen.

4.1.3.6.184 Felddeklarationen sind im Typ interface nicht erlaubt (E2209)

Eine Schnittstelle enthält unzulässigerweise Felddefinitionen.

```
program Produce;
type
  IBaseIntf = interface
    FVar : Integer;
    property Value : Integer read FVar write FVar;
  end;

begin
end.
```

In diesem Beispiel wird versucht, einer Eigenschaft einen Wert zuzuweisen. Das funktioniert aber nicht, weil Schnittstellen keine

Felder enthalten dürfen.

```
program Solve;
  IBaseIntf = interface
    function Reader : Integer;
    procedure Writer(a : Integer);
    property Value : Integer read Reader write Writer;
  end;

begin
end.
```

Eine elegante Lösung für dieses Problem besteht in der Deklaration von Abruf- und Zuweisungsprozeduren für die Eigenschaft. In diesem Fall muss jede Klasse, welche die Schnittstelle implementiert, eine Methode bereitstellen, die für den Zugriff auf die Daten der Klasse eingesetzt wird.

4.1.3.6.185 Chmod-Fehler für '%s' (x2044)

Die Zugriffsrechte wurden für eine Datei nicht richtig festgelegt. Weitere Informationen finden Sie in der Dokumentation zu chmod.

4.1.3.6.186 Fehler beim Schließen von '%s' (x2043)

Der Compiler ist beim Schließen einer Eingabe- oder Ausgabedatei auf einen Fehler gestoßen.

Dieser Fall sollte nur selten eintreten. Wenn er eintritt, ist die Ursache mit größter Wahrscheinlichkeit eine voller oder fehlerhafter Datenträger..

4.1.3.6.187 Ausgabedatei '%s' kann nicht erstellt werden (F2039)

Der Compiler konnte die Ausgabedatei nicht erstellen. Dabei kann es sich um eine compilierte Unit-Datei (.dcu), eine ausführbare Datei, eine Map-Datei oder eine Objektdatei handeln.

Die häufigste Fehlerursache ist ein nicht vorhandenes Ausgabeverzeichnis oder ein schreibgeschützter Datenträger.

4.1.3.6.188 Falsches Dateiformat: '%s' (x2141)

Die Compiler-Zustandsdatei wurde beschädigt. Es ist nicht möglich, den vorherigen Compiler-Zustand zu laden.

Löschen Sie die beschädigte Datei.

4.1.3.6.189 Dateiname zu lang (übersteigt %d Zeichen) (E2288)

Ein in den Compiler-Optionen festgelegter Dateipfad übersteigt die Dateipufferlänge des Compilers.

4.1.3.6.190 Datei nicht gefunden: '%s' (x1026)

Diese Fehlermeldung tritt auf, wenn der Compiler eine Eingabedatei nicht finden kann. Es kann sich hierbei um eine Quelldatei, eine compilierte Unit-Datei (.dcu)-Datei), eine Include-Datei, eine Objektdatei oder eine Ressourcendatei handeln.

Überprüfen Sie die Schreibweise der Bezeichnung und den entsprechenden Suchpfad.

```
program Produce;
uses Borland.Vcl.SysUtilss;      (*--- Hier die Fehlermeldung*)
begin
```

```

end.
program Solve;
uses Borland.Vcl.SysUtils;      (*Korrigierter Schreibfehler*)
begin
end.

```

Bei einer .dcu1-Datei ist eine wahrscheinlich Ursache für diese Meldung ein vergessener Unit-/Bibliothekspfad für den Compiler. Die einzige Lösung ist, sicherzustellen, dass die Unit über den Bibliothekspfad gefunden werden kann.

4.1.3.6.191 Unit nicht gefunden: '%s' oder binäre Äquivalente (%s) (F1027)

Diese Fehlermeldung tritt auf, wenn der Compiler eine referenzierte Unit-Datei (.dcu1) nicht finden kann.

Überprüfen Sie die Schreibweise der referenzierten Datei und den entsprechenden Suchpfad.

4

4.1.3.6.192 Fehler beim Lesen von '%s' (x2041)

Der Compiler ist in einer Eingabedatei auf einen Lesefehler gestoßen.

Dieser Fall sollte niemals eintreten – falls doch, ist die Ursache mit größter Wahrscheinlichkeit in beschädigten Daten zu finden.

4.1.3.6.193 Fehler beim Suchen von '%s' (F2040)

Der Compiler ist in einer Eingabe- oder Ausgabedatei auf einen Suchfehler gestoßen.

Dieser Fall sollte niemals eintreten – falls doch, ist die Ursache mit größter Wahrscheinlichkeit in beschädigten Daten zu finden.

4.1.3.6.194 Dateityp ist hier nicht zulässig (E2002)

Dateitypen sind als Wertparameter und als Basistyp eines Dateityps selber nicht zulässig. Außerdem dürfen sie nicht als Rückgabewerte von Funktionen eingesetzt werden, und sie können nicht zugewiesen werden – diese Fehler führen allerdings zu anderen Fehlermeldungen.

```

program Produce;

procedure WriteInteger(T: Text; I: Integer);
begin
  Writeln(T, I);
end;

begin
end.

```

In diesem Beispiel liegt das Problem darin, dass T ein Wertparameter vom Typ Text ist, welcher wiederum ein Dateityp ist. Denken Sie daran: Ganz gleich, was zu einem Wertparameter geschrieben wird – auf die Kopie der Variable beim Aufrufenden wird kein Einfluss genommen. Die Deklaration einer Datei als Wertparameter hat daher wenig Sinn.

```

program Solve;

procedure WriteInteger(var T: Text; I: Integer);
begin
  Writeln(T, I);
end;

begin
end.

```

Die Deklaration des Parameters als var-Parameter beseitigt das Problem.

4.1.3.6.195 Fehler beim Schreiben von '%s' (x2042)

Der Compiler ist während des Schreibens in eine Ausgabedatei auf einen Schreibfehler gestoßen.

Aller Wahrscheinlichkeit nach ist die Kapazität des Datenträgers erschöpft.

4.1.3.6.196 Final-Methoden müssen virtuell oder dynamisch sein (E2351)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.197 Typ '%s' benötigt Finalisierung - im Dateityp nicht erlaubt (E2155)

Bestimmte Typen werden vom Compiler intern auf besondere Weise behandelt: Sie müssen zuerst ordnungsgemäß abgeschlossen werden, damit gegebenenfalls die gesamten von ihnen zurzeit belegten Ressourcen freigegeben werden. Da der Compiler nicht feststellen kann, welcher Typ während der Programmausführung tatsächlich im Variantenabschnitt eines Datensatzes gespeichert ist, kann nicht garantiert werden, dass diese speziellen Datentypen ordnungsgemäß abgeschlossen werden.

```
program Produce;

type
  Data = record
    name : string;
  end;

var
  inFile : file of Data;

begin
end.

String ist einer der Typen, für die ein besonderer Abschluss durchgeführt werden muss, und kann daher nicht in einem Dateityp gespeichert werden.

program Solve;

type
  Data = record
    name : array[1..25] of Char;
  end;

var
  inFile : file of Data;

begin
end.
```

Eine einfache Lösung für den Typ String ist die Neudeklaration des Typs als Zeichen-Array. In anderen Fällen, in denen ein besonderer Abschluss durchgeführt werden muss, wird es immer schwieriger, eine binäre Dateistruktur mit Standard-Pascal-Merkmalen wie "file of" aufrechtzuerhalten. In diesen Situationen ist es wahrscheinlich einfacher, spezialisierte Datei-E/A-Routinen zu schreiben.

4.1.3.6.198 Typ '%s' benötigt Finalisierung - nicht im Variant-Record erlaubt (E2154)

Bestimmte Typen werden vom Compiler intern auf besondere Weise behandelt: Sie müssen zuerst ordnungsgemäß abgeschlossen werden, damit gegebenenfalls die gesamten von ihnen zurzeit belegten Ressourcen freigegeben werden. Da der

Compiler nicht feststellen kann, welcher Typ während der Programmausführung tatsächlich im Variantenabschnitt eines Datensatzes gespeichert ist, kann nicht garantiert werden, dass diese speziellen Datentypen ordnungsgemäß abgeschlossen werden.

```
program Produce;

type
  Data = record
    case kind:Char of
      'A': (str : String);
  end;
```

```
begin
end.
```

String ist einer der Typen, die eine besondere Behandlung vom Compiler erfordern, um die belegten Ressourcen ordnungsgemäß freigeben zu können. Aus diesem Grunde ist die Benutzung von String in einem Variant-Abschnitt nicht zulässig.

```
program Solve;

type
  Data = record
    str : String;
  end;
```

```
begin
end.
```

Eine Lösung für diesen Fehler ist, alle betreffenden Deklarationen aus dem Variant-Abschnitt zu entfernen. Eine andere Möglichkeit wäre, Zeigertypen zu verwenden (beispielsweise ^String) und den Speicher selbst zu verwalten.

4.1.3.6.199 16-Bit Fixup in Objektdatei '%s' entdeckt (E2103)

Der Compiler hat in einem der Objektmodule, die mit der Direktive \$L zu Ihrem Programm gelinkt wurden, ein 16-Bit-Fixup entdeckt. In gelinkten Objektmodulen werden jedoch nur 32-Bit-Fixups unterstützt.

Stellen Sie sicher, dass das gelinkte Objektmodul ein 32-Bit-Modul ist.

4.1.3.6.200 Ungenügende Forward- oder External-Deklaration: '%s' (E2065)

Diese Fehlermeldung tritt auf, wenn eine forward- oder external-Deklaration einer Prozedur bzw. Funktion oder eine Methodendeklaration in einem Klassen- bzw. Objekttyp vorhanden ist, aber die Prozedur, Funktion oder Methode nicht definiert wird.

Möglicherweise fehlt die Definition, oder ihr Name wurde falsch geschrieben.

Beachten Sie, dass eine Prozedur- oder Funktionsdeklaration im interface-Abschnitt einer Unit einer forward-Deklaration entspricht. Die zugehörige Implementierung (der Quelltext der Routine) muss dann im implementation-Abschnitt erstellt werden.

Ebenso entspricht eine Methodendeklaration in einem Klassen- oder Objekttyp einer forward-Deklaration.

```
program Produce;

type
  TMyClass = class
    constructor Create;
  end;

  function Sum(const a: array of Double): Double; forward;
```

```

function Summ(const a: array of Double): Double;
var
  i : Integer;
begin
  Result := 0.0;
  for i:= 0 to High(a) do
    Result := Result + a[i];
end;

begin
end.

```

Die Definition von Sum hat einen leicht zu erkennenden Schreibfehler.

```

program Solve;

type
  TMyClass = class
    constructor Create;
  end;

constructor TMyClass.Create;
begin
end;

function Sum(const a: array of Double): Double; forward;

function Sum(const a: array of Double): Double;
var
  i : Integer;
begin
  Result := 0.0;
  for i:= 0 to High(a) do
    Result := Result + a[i];
end;

begin
end.

```

Lösung: Vergewissern Sie sich, dass die Definitionen Ihrer Prozeduren, Funktionen und Methoden vorhanden und richtig geschrieben sind.

4.1.3.6.201 Deklaration von '%s' unterscheidet sich von vorheriger Deklaration (E2037)

Diese Fehlermeldung tritt auf, wenn die Deklaration einer Prozedur, einer Funktion, einer Methode, eines Konstruktors oder eines Destruktors nicht mit der bisherigen (Forward-)Deklaration übereinstimmt.

Diese Fehlermeldung tritt außerdem auf, wenn Sie versuchen, eine virtuelle Methode zu überschreiben, die überschreibende Methode jedoch eine andere Parameterliste, andere Aufrufkonventionen usw. aufweist.

```

program Produce;

type
  MyClass = class
    procedure Proc(Idx: Integer);
    function Func: Integer;
    procedure Load(const Name: string);
    procedure Perform(Flag: Boolean);
    constructor Create;
    destructor Destroy(Msg: string); override; (*<-- Hier die Fehlermeldung*)
    class function NewInstance: MyClass; override; (*<-- Hier die Fehlermeldung*)
  end;

```

```

procedure MyClass.Proc(Index: Integer);           (*<-- Hier die Fehlermeldung*)
begin
end;

function MyClass.Func: Longint;                  (*<-- Hier die Fehlermeldung*)
begin
end;

procedure MyClass.Load(Name: string);            (*<-- Hier die Fehlermeldung*)
begin
end;

procedure MyClass.Perform(Flag: Boolean);         (*<-- Hier die Fehlermeldung*)
cdecl
begin
end;

procedure MyClass.Create;                         (*<-- Hier die Fehlermeldung*)
begin
end;

function MyClass.NewInstance: MyClass;           (*<-- Hier die Fehlermeldung*)
begin
end;

begin
end.

```

Wie Sie sehen können, gibt es eine ganze Reihe von Ursachen für diese Fehlermeldung.

```

program Solve;

type
  MyClass = class
    procedure Proc(Idx: Integer);
    function Func: Integer;
    procedure Load(const Name: string);
    procedure Perform(Flag: Boolean);
    constructor Create;
    destructor Destroy; override;           (*Keine Parameter*)
    class function NewInstance: TObject; override; (*Ergebnistyp *)
  end;

  procedure MyClass.Proc(Idx: Integer);        (*Parametername *)
begin
end;

  function MyClass.Func: Integer;              (*Ergebnistyp *)
begin
end;

  procedure MyClass.Load(const Name: string);  (*Parameterart *)
begin
end;

  procedure MyClass.Perform(Flag: Boolean);    (*Aufrufkonvention*)
begin
end;

  constructor MyClass.Create;                 (*Konstruktor*)
begin
end;

  class function MyClass.NewInstance: TObject;(*Klassenfunktion*)
begin
end;

begin

```

end.

Sie müssen die 'vorherigen Deklaration' sorgfältig mit der vergleichen, die den Fehler ausgelöst hat, um Unterschiede zwischen den beiden Deklarationen feststellen zu können.

4.1.3.6.202 FOR-Schleifenvariable '%s' kann nach Durchlauf undefiniert sein (W1037)

Diese Warnung wird angezeigt, wenn die Steuervariable einer for-Schleife nach der Schleife verwendet wird.

Sie können sich nur auf den letzten Wert eines for-Schleifenzählers verlassen, wenn die Schleife mit einer goto- oder exit-Anweisung verlassen wird.

Der Grund für diese Einschränkung ist, dass der Compiler dadurch sehr effizienten Code für die for-Schleife erzeugen kann.

```
program Produce;
(*$WARNINGS ON*)

function Search(const A: array of Integer; Value: Integer): Integer;
begin
  for Result := 0 to High(A) do
    if A[Result] = Value then
      break;
end;

const
  A: array [0..9] of Integer = (1,2,3,4,5,6,7,8,9,10);

begin
  Writeln( Search(A,11) );
end.
```

In diesem Beispiel wird die Variable Result implizit nach der Schleife verwendet. Da sie aber undefiniert ist, wenn der Wert nicht gefunden wird, gibt der Compiler eine Warnung aus.

```
program Solve;
(*$WARNINGS ON*)

function Search(const A: array of Integer; Value: Integer): Integer;
begin
  for Result := 0 to High(A) do
    if A[Result] = Value then
      exit;
  Result := High(a)+1;
end;

const
  A: array [0..9] of Integer = (1,2,3,4,5,6,7,8,9,10);

begin
  Writeln( Search(A,11) );
end.
```

Die Lösung besteht darin, der Steuervariablen den gewünschten Wert für den Fall zuzuweisen, dass die Schleife nicht vorzeitig beendet wird.

4.1.3.6.203 FOR-Schleifenvariable '%s' kann nicht als Var-Parameter übergegeben werden (W1015)

Sie haben versucht, die Steuervariable einer for-Schleife an eine Prozedur oder Funktion zu übergeben, die einen var-Parameter erwartet. Dies ist eine Warnung, weil die Variable in der Routine geändert werden kann und dadurch die Semantik der

for-Schleife verändert wird.

```
program Produce;

procedure p1(var x : Integer);
begin
end;

procedure p0;
var
  i : Integer;
begin
  for i := 0 to 1000 do
    p1(i);
end;

begin
end.
```

In diesem Beispiel wird der Schleifenzähler *i* an eine Prozedur übergeben, die einen var-Parameter erwartet. Dies ist die Hauptursache der Warnung.

```
program Solve;
procedure p1(x : Integer);
begin
end;

procedure p0;
var
  i : Integer;
begin
  i := 0;
  while i <= 1000 do
    p1(i);
end;

begin
end.
```

Die einfachste Lösung des Problems besteht darin, das Argument als by-value-Parameter zu deklarieren. Sie müssen dann aber sicherstellen, dass sich diese Änderung nicht auf andere Abschnitte des Quelltextes auswirkt. Es ist daher vorzuziehen, wie im obigen Programm die for- durch eine entsprechende while-Schleife zu ersetzen.

4.1.3.6.204 FOR-Schleifenvariable muss von ordinalem Typ sein (E2032)

Die Steuervariable einer for-Schleife muss vom Typ Boolean, Char, WideChar oder Integer bzw.

```
program Produce;
var
  x: Real;
begin (*Plot sine wave*)
  for x := 0 to 2*pi/0.2 do
    Writeln( '*' : Round((Sin(x*0.2) + 1)*20) + 1 );
end.
```

In diesem Beispiel wird eine Variable vom Typ Real als Steuervariable für die for-Schleife verwendet; dies ist nicht zulässig.

```
program Solve;
var
  x: Integer;
begin (*Plot sine wave*)
  for x := 0 to Round(2*pi/0.2) do
    Writeln( '*' : Round((Sin(x*0.2) + 1)*20) + 1 );
end.
```

Stattdessen muss der Ordinaltyp Integer verwendet werden.

Diese Fehlermeldung wird angezeigt, wenn eine Steuervariable des Typs Int64 oder Variant in einer for-Schleife verwendet wird. Sie können dieses Compiler-Problem beheben, indem Sie die for-Schleife durch eine while-Schleife ersetzen.

4.1.3.6.205 FOR-Schleifenvariable muss eine einfache lokale Variable sein (x1019)

Diese Fehlermeldung wird angezeigt, wenn die Steuervariable einer for-Anweisung keine einfache Variable ist (sondern beispielsweise eine Komponente eines Datensatzes), und wenn sie nicht lokal zu der Prozedur ist, die die for-Anweisung enthält.

Aus Gründen der Abwärtskompatibilität ist es zulässig, eine globale Variable als Steuervariable zu verwenden – der Compiler gibt in diesem Fall eine Warnmeldung aus. Beachten Sie, dass mit Verwendung einer lokalen Variable außerdem ein leistungsfähigerer Programmcode erzeugt wird.

```
program Produce;

var
  I: Integer;
  A: array[0..9] of Integer;

procedure Init;
begin
  for I := Low(A) to High(A) do (*<-- Hier die Warnung*)
    A[I] := 0;
end;

begin
  Init;
end.

program Solve;
var
  A: array[0..9] of Integer;

procedure Init;
var
  I: Integer;
begin
  for I := Low(A) to High(A) do
    A[I] := 0;
end;

begin
  Init;
end.
```

4.1.3.6.206 Text hinter dem abschließenden 'END.' wird vom Compiler ignoriert (W1011)

Diese Warnmeldung wird angezeigt, wenn sich nach dem abschließenden end und dem Punkt, der das logische Ende des Programms bildet, weiterer Quelltext befindet. Wahrscheinlich ist die Verschachtelung von begin..end inkonsistent (irgendwo ist ein "end" zu viel). Überprüfen Sie, ob der Quelltext wirklich vom Compiler ignoriert werden sollte – möglicherweise ist er recht bedeutend.

```
program Produce;

begin
end.

Text here is ignored by Delphi 16-bit - Delphi 32-bit or Kylix gives a warning.
```

```
program Solve;  
begin  
end.
```

4.1.3.6.207 'GOTO %s' führt in oder aus einer TRY-Anweisung (E2127)

Eine goto-Anweisung kann nicht in eine Anweisung der Exception-Behandlung oder aus ihr heraus führen.

```
program Produce;  
label 1, 2;  
begin  
  goto 1;  
  try  
  1:  
    except  
      goto 2;  
    end;  
  2:  
end.
```

4

Die beiden goto-Anweisungen im obigen Code sind fehlerhaft. Es ist nicht möglich, auf diese Weise in Exception-Behandlungsblöcke einzutreten oder sie zu verlassen.

Die beste Lösung dieses Problems besteht darin, goto-Anweisungen generell zu vermeiden. Ist dies nicht möglich, müssen Sie Ihr Programm genau untersuchen, um einen fehlerfreien Ablauf zu erreichen.

4.1.3.6.208 Eine unterstützende Klasse darf keinen Destruktor einführen (E2295)

Unterstützende Klassen dürfen keine Destruktoren deklarieren.

4.1.3.6.209 Eine benötigte Bibliothekshilfsfunktion wurde vom Linker eliminiert (%s) (E2172)

Der integrierte Debugger versucht, einige der Compiler-Hilfsfunktionen für die Ausführung der angeforderten Auswertung zu benutzen. Der Linker hat auf der anderen Seite festgestellt, dass die Hilfsfunktion nicht vom Programm benutzt wurde, und hat sie daher nicht mit dem Programm verknüpft.

1. Erstellen Sie eine neue Anwendung.
2. Platzieren Sie eine Schaltfläche im Formular.
3. Klicken Sie doppelt auf die Schaltfläche aus, um zur Klick-Methode zu gelangen.
4. Fügen Sie eine globale Variable 'v' vom Typ string zum Interface-Abschnitt hinzu.
5. Fügen Sie eine globale Variable 'p' vom Typ PChar zum Interface-Abschnitt hinzu.

Die Klick-Methode sollte folgendermaßen aussehen:

1. procedure TForm1.Button1Click(Sender: TObject); begin v := 'Initialized'; p := NIL; v := 'Abid'; end;
2. Setzen Sie einen Haltepunkt bei der zweiten Zuweisung zu 'v'.
3. Compilieren Sie die Anwendung, und führen Sie sie aus.
4. Klicken Sie auf die Schaltfläche.
5. Wenn der Haltepunkt erreicht wird, öffnen Sie den Auswerter (Start/Auswerten/Ändern).

6. Werten Sie V aus.
 7. Setzen Sie den Cursor in das Feld "Neuer Wert".
 8. Geben Sie 'p' ein.
 9. Wählen Sie Ändern.

Der Compiler verwendet eine spezielle Funktion, um ein PChar in einen String zu kopieren. Falls diese spezielle Funktion vom Programm nicht benutzt wird, wird sie zur Verringerung der Größe der erzeugten ausführbaren Datei nicht mit ihr verknüpft. In diesem Fall findet keine Zuweisung eines PChar zu einem String statt, so dass er vom Linker gelöscht wird.

```
4
procedure TForm1.Button1Click(Sender: TObject);
begin
  v := 'Initialized';
  p := NIL;
  v := 'Abid';
  v := p;
end;
```

Durch Einbau einer zusätzlichen Zuweisung eines PChar zu einem String wird sichergestellt, dass der Linker die gewünschte Prozedur in das Programm aufnimmt. Wenn Sie während einer Debugging-Sitzung auf diesen Fehler stoßen, deutet dies darauf hin, dass Sie ein Sprachen- oder Umgebungsmerkmal verwenden, das im ursprünglichen Programm nicht gebraucht wurde.

4.1.3.6.210 Methode '%s' verbirgt virtuelle Methode vom Basistyp '%s' (W1010)

Sie haben eine Methode deklariert, die dieselbe Bezeichnung wie eine virtuelle Methode in der Basisklasse hat. Diese neue Methode ist keine virtuelle Methode und wird den Zugriff auch die gleichnamige Basismethode verbergen.

```
program Produce;

type
  Base = class
    procedure VirtuMethod; virtual;
    procedure VirtuMethod2; virtual;
  end;

  Derived = class (Base)
    procedure VirtuMethod;
    procedure VirtuMethod2;
  end;

procedure Base.VirtuMethod;
begin
end;

procedure Base.VirtuMethod2;
begin
end;

procedure Derived.VirtuMethod;
begin
end;

procedure Derived.VirtuMethod2;
begin
end;

begin
end.
```

Beide in der Definition von Derived deklarierten Methoden verbergen die virtuellen Funktionen mit derselben Bezeichnung in der Basisklasse.

```
program Solve;
```

```

type
  Base = class
    procedure VirtuMethod; virtual;
    procedure VirtuMethod2; virtual;
  end;

  Derived = class (Base)
    procedure VirtuMethod; override;
    procedure Virtu2Method;
  end;

procedure Base.VirtuMethod;
begin
end;

procedure Base.VirtuMethod2;
begin
end;

procedure Derived.VirtuMethod;
begin
end;

procedure Derived.Virtu2Method;
begin
end;

begin
end.

```

Zur Beseitigung dieses Fehlers gibt es drei Möglichkeiten:

Zum einen können Sie override angeben und auf diese Weise die Prozedur der abgeleiteten Klasse ebenfalls virtuell machen, sodass geerbte Aufrufe sich immer noch auf die ursprüngliche Prozedur beziehen können.

Zweitens könnten Sie auch die Bezeichnung der Prozedur in der Deklaration der abgeleiteten Klasse ändern. In diesem Beispiel werden beide Methoden gezeigt.

Drittens können Sie die Direktive reintroduce in die Deklaration der Prozedur einfügen, um die Warnung für diese Methode zu unterdrücken.

4.1.3.6.211 Redeklaration von '%s' verbirgt ein Mitglied (Member) in der Basisklasse (W1009)

In einer Klasse wurde eine Eigenschaft erstellt, die dieselbe Bezeichnung wie eine Variable in einer der Basisklassen hat. Eine mögliche, aber nicht immer offensichtliche Ursache für diesen Fehler liegt darin, dass eine neue Version der Basisklassen-Hierarchie installiert wurde, die neue Mitgliedervariablen aufweist, deren Bezeichnungen mit den Bezeichnungen Ihrer Eigenschaften identisch sind. -W

```

(*$WARNINGS ON*)
program Produce;

type
  Base = class
    v : integer;
  end;

  Derived = class (Base)
    ch : char;
    property v : char read ch write ch;
  end;

```

```
begin
end.
```

Derived.v überschreibt, d. h. verbirgt, Base.v; der Zugriff auf Base.v ist ohne Typumwandlung aus keiner Variablen vom Typ Derived möglich.

```
(*$WARNINGS ON*)
program Solve;
type
  Base = class
    v : integer;
  end;

  Derived = class (Base)
    ch : char;
    property chV : char read ch write ch;
  end;

begin
end.
```

Der Fehler ist nach Änderung der Bezeichnung der Eigenschaft in der abgeleiteten Klasse behoben.

4.1.3.6.212 HIGH kann nicht auf lange Stringtypen angewendet werden (E2198)

Es ist nicht möglich, die Standardfunktion HIGH mit langen Strings zu verwenden. Diese Funktion kann aber mit den kurzen Strings älterer Versionen eingesetzt werden.

Da lange Strings ihre Größe dynamisch verändern, existiert keine Entsprechung, die anstelle der Funktion HIGH verwendet werden könnte.

Dieser Fehler kann bei der Portierung einer 16-Bit-Anwendung auftreten, da in dieser Situation nur ein kurzer String verfügbar ist. Wurde der Fehler während einer derartigen Portierung erzeugt, können Sie die langen Strings mit der Kommandozeilenoption \$H oder mit der Direktive \$LONGSTRINGS deaktivieren.

Wenn Sie die Funktion HIGH auf einen String-Parameter angewendet haben, aber trotzdem lange Strings benutzen wollen, ändern Sie den Parametertyp in openstring.

```
program Produce;
var
  i : Integer;
  s: String;

begin
  s := 'Hello Developers of the World';
  i := HIGH(s);
end.
```

In diesem Beispiel wurde versucht, die Standardfunktion HIGH für eine lange String-Variable anzuwenden. Dies ist aber nicht möglich.

```
(*$LONGSTRINGS OFF*)
program Solve;
var
  i : Integer;
  s: String;

begin
  s := 'Hello Developers of the World';
  i := HIGH(s);
end.
```

Wenn Parameter mit langen Strings deaktiviert werden, ist die Verwendung von HIGH für eine String-Variable möglich.

4.1.3.6.213 \$HPPEMIT '%s' wird ignoriert (W1034)

Die Anweisung \$HPPEMIT darf nur im Unit-Header vorkommen.

4.1.3.6.214 Integer und HResult werden ausgetauscht (x1008)

Integer, Longint und HRESULT sind in Delphi kompatible Typen. In C++ sind sie jedoch nicht kompatibel und erzeugen unterschiedlich verkürzte C++ Parameternamen. Diese Meldung soll sicherstellen, dass es beim Linken von Objektdateien, die Sie mit dem Delphi-Compiler erzeugt haben, keine Probleme gibt. Wenn Sie eine Objektdatei compilieren, ergibt sich daraus ein Fehler. Andernfalls ist die Meldung als Warnung zu betrachten.

```
program Produce;
  uses Windows;

  type
    I0 = interface (IUnknown)
      procedure p0(var x : Integer);
    end;

    C0 = class (TInterfacedObject, I0)
      procedure p0(var x : HRESULT);
    end;

    procedure C0.p0(var x : HRESULT);
  begin
  end;

begin
end.
```

In diesem Beispiel werden die Schnittstelle und die Klassenmethoden unterschiedlich deklariert. Beide Methoden sind in Delphi äquivalent, nicht jedoch in C++.

```
program Solve;
  uses Windows;

  type
    I0 = interface (IUnknown)
      procedure p0(var x : Integer);
    end;

    C0 = class (TInterfacedObject, I0)
      procedure p0(var x : Integer);
    end;

    procedure C0.p0(var x : Integer);
  begin
  end;

begin
end.
```

Die einfachste Lösung ist, die unter class deklarierten Methoden mit den unter interface deklarierten Methoden abzuleichen, sodass sie identisch sind.

4.1.3.6.215 W1000 Symbol '%s' wird abgelehnt (W1000)

Das Symbol ist (mit der Hinweisdirektive **deprecated**) als nicht mehr aktuell gekennzeichnet und nur aus Kompatibilitätsgründen vorhanden. Verwenden Sie möglichst ein anderes Symbol in Ihrem Quelltext.

Mit der Compiler-Direktive **\$WARN SYMBOL_DEPRECATED** ON/OFF können alle entsprechenden Warnungen für Symbole in der aktuellen Unit aktiviert oder deaktiviert werden.

4.1.3.6.216 Bezeichner erwartet (E2372)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.217 Symbol '%s' ist experimental (W1003)

Eine "experimental" Direktive wurde für einen Bezeichner verwendet. "Experimental" gibt an, dass eine Klasse oder Unit unvollständig oder nicht ausreichend getestet ist.

4.1.3.6.218 Symbol '%s' ist bibliotheksspezifisch (W1001)

Das Symbol ist (mit der Hinweisdirektive **library**) als nicht in allen Bibliotheken verfügbar gekennzeichnet. Wenn Sie andere Bibliotheken verwenden, kann dies zu Problemen führen.

Mit der Compiler-Direktive **\$WARN SYMBOL_LIBRARY** ON/OFF können alle entsprechenden Warnungen für Symbole in der aktuellen Unit aktiviert oder deaktiviert werden.

4.1.3.6.219 Symbo '%s' ist plattformspezifisch (W1002)

Das Symbol ist (mit der Hinweisdirektive **platform**) als nicht auf allen Plattformen verfügbar gekennzeichnet. Wenn Sie plattformübergreifende Anwendungen erstellen, kann dies zu Problemen führen.

Mit der Compiler-Direktive **\$WARN SYMBOL_PLATFORM** ON/OFF können alle entsprechenden Warnungen für Symbole in der aktuellen Unit aktiviert oder deaktiviert werden.

4.1.3.6.220 Bezeichner redeklariert: '%s' (E2004)

Der angegebene Bezeichner wurde bereits für diesen Gültigkeitsbereich definiert – Sie versuchen, seine Bezeichnung für etwas anderes wiederzuverwenden.

```
program Tests;
var
  Tests: Integer;
begin
end.
```

Hier ist die Bezeichnung des Programms dieselbe wie die der Variablen – wir müssen eine der Bezeichnungen ändern, um den Compiler zufrieden zu stellen.

```
program Tests;
var
  TestCnt: Integer;
begin
end.
```

4.1.3.6.221 Undefinierter Bezeichner: '%s' (E2003)

Der Compiler konnte den angegebenen Bezeichner nicht finden – aller Wahrscheinlichkeit nach wurde er während der Deklaration bzw. während der Benutzung falsch geschrieben. Er könnte zu einer anderen Unit gehören, die keine uses-Anweisung aufweist.

```
program Produce;
var
  Counter: Integer;
begin
  Count := 0;
  Inc(Count);
  Writeln(Count);
end.
```

In diesem Beispiel wurde die Variable als "Counter" deklariert, aber als "Count" benutzt. Die Lösung liegt darin, entweder die Deklaration zu ändern oder die Stellen zu berichtigen, an denen die Variable benutzt wird.

```
program Solve;
var
  Count: Integer;
begin
  Count := 0;
  Inc(Count);
  Writeln(Count);
end.
```

In diesem Beispiel haben wir uns dafür entschieden, die Deklaration zu ändern – das war ein geringerer Arbeitsaufwand.

4.1.3.6.222 Es kann nur entweder IID oder GuidAttribute angegeben werden (E2427)

Die GUID oder IID Ihres Interface kann durch eckige Klammern am Beginn der Interface-Deklaration oder durch ein .NET-Attribut vor der Interface-Deklaration angegeben werden. Sie können eine der beiden Methoden für die GUID- oder IID-Deklaration verwenden, aber nicht beide im selben Typ.

4.1.3.6.223 Ungültiger Typ im OLE-Automatisierungsbereich: '%s' (E2176)

<Element> ist kein im Abschnitt automated zulässiger Typ. In automated-Abschnitten ist nur eine begrenzte Auswahl der gültigen Delphi-Typen zulässig.

```
program Produce;

type
  Base = class
    function GetC: Char;
    procedure SetC(c: Char);
  automated
    property Ch: Char read GetC write SetC dispid 151;
  end;

  procedure Base.SetC(c: Char);
begin
end;

  function Base.GetC: Char;
begin
  GetC := '!';
end;

begin
end.
```

Da der Zeichentyp nicht im Abschnitt automated zulässig ist, löst die Deklaration von Ch bei der Compilierung einen Fehler aus.

```
program Solve;
```

```
type
  Base = class
```

```

        function GetC : String;
        procedure SetC(c : String);
automated
        property Ch : String read GetC write SetC dispid 151;
end;

procedure Base.SetC(c : String);
begin
end;

function Base.GetC : String;
begin GetC := '!';
end;

begin
end.

```

Es gibt zwei Lösungen für dieses Problem. Zum einen kann die betreffende Deklaration aus dem Abschnitt automated herausgebracht werden. Zum anderen kann der betreffende Typ in einen Typ geändert werden, der in einem Abschnitt automated zulässig ist.

4.1.3.6.224 Überschriebene automatisierte virtuelle Methode '%s' kann kein dispid definieren (E2185)

Die während der Deklaration der ursprünglichen virtuellen automated-Prozedur deklarierte dispid-Anweisung muss von allen überschreibenden Prozeduren in den abgeleiteten Klassen benutzt werden.

```

program Produce;

type
  Base = class
  automated
    procedure Automatic; virtual; dispid 151;
  end;

  Derived = class (Base)
  automated
    procedure Automatic; override; dispid 152;
  end;

procedure Base.Automatic;
begin
end;

procedure Derived.Automatic;
begin
end;

begin
end.

```

Die überschreibende Deklaration von Base.Automatic in Derived (Derived.Automatic) versucht unzulässigerweise, einen anderen DispId für die Prozedur zu definieren.

```

program Solve;

type
  Base = class
  automated
    procedure Automatic; virtual; dispid 151;
  end;

```

```

Derived = class (Base)
automated
  procedure Automatic; override;
end;

procedure Base.Automatic;
begin
end;

procedure Derived.Automatic;
begin
end;

begin
end.

```

Die Compilierung des Programms wird durchgeführt, wenn die betreffende `dispid`-Anweisung entfernt wird.

4.1.3.6.225 Ungültige Referenz auf Symbol '%s' in Objektdatei '%s' (E2068)

Diese Fehlermeldung wird angezeigt, wenn eine mit der Direktive `$L` oder `$LINK` eingebundene Objektdatei einen Verweis auf ein Delphi-Symbol enthält, das keine Prozedur, Funktion, Variable, typisierte Konstante oder Thread-lokale Variable ist.

4.1.3.6.226 Ungültiger Botschaftsmethoden-Index (E2139)

Sie haben für den Botschaftsindex einen Wert ≤ 0 angegeben.

```

program Produce;

type
  Base = class
    procedure Dynamo(VAR x : Integer); message -151;
  end;

procedure Base.Dynamo(VAR x : Integer);
begin
end;

begin
end.

```

In diesem Beispiel ist der Botschaftsindex `-151` nicht erlaubt.

```

program Solve;

type
  Base = class
    procedure Dynamo(VAR x : Integer); message 151;
  end;

procedure Base.Dynamo(VAR x : Integer);
begin
end;

begin
end.

```

Botschaftsindizes müssen immer ≥ 1 sein.

4.1.3.6.227 \$DESIGNONLY und \$RUNONLY sind nur in einer Package-Unit erlaubt (E2224)

Der Compiler hat in einer Quelldatei, bei der es sich nicht um ein Package handelt, entweder die Direktive \$DESIGNONLY oder die Direktive \$RUNONLY gefunden. Diese Direktiven beeinflussen die Art und Weise, in der die Entwicklungsumgebung eine Package-Datei behandelt. Sie können deshalb auch nur in Package-Quelldateien angegeben werden.

4.1.3.6.228 %s-Abschnitt ist nur in Klassentypen gültig (E2184)

Interfaces und Records dürfen keine published-Abschnitte enthalten.

Records dürfen keine protected-Abschnitte enthalten.

4.1.3.6.229 Ungültiges Zeichen in Eingabedatei: '%s' (%s) (E2038)

Der Compiler hat ein Zeichen gefunden, das in Delphi-Programmen nicht zulässig ist.

Diese Fehlermeldung wird am häufigsten durch Fehler im Zusammenhang mit Stringkonstanten und Kommentaren ausgelöst.

```
program Produce;
```

```
begin
  Writeln("Hello world!"); (*<-- Hier die Fehlermeldung*)
end.
```

Hier hat ein Programmierer einen String fälschlicherweise in doppelte Anführungszeichen gesetzt (C++ Stil).

```
program Solve;
```

```
begin
  Writeln('Hello world!'); (*In Delphi sind einfache Anführungszeichen erforderlich*)
end.
```

Die Lösung liegt darin, einfache Anführungszeichen zu benutzen. Im Allgemeinen müssen Sie das unzulässige Zeichen löschen.

4.1.3.6.230 '%s' Anweisung im OLE Automatisierungsbereich nicht erlaubt (E2182)

index, stored, default und nodefault sind in OLE-Automatisierungsabschnitten nicht zulässig.

```
program Produce;

type
  Base = class
    v : integer;
    procedure setV(x : integer);
    function getV : integer;
    automated
    property Value : integer read getV write setV nodefault;
  end;

  procedure Base.setV(x : integer);
begin v := x;
end;

  function Base.getV : integer;
begin getV := v;
end;
```

```
begin
end.
```

Die Verwendung einer `nodefault`-Klausel mit einer automatisierten Eigenschaft ist nicht zulässig.

```
program Solve;

type
  Base = class
    v : integer;
    procedure setV(x : integer);
    function getV : integer;
    automated
      property Value : integer read getV write setV;
    end;

    procedure Base.setV(x : integer);
    begin v := x;
    end;

    function Base.getV : integer;
    begin getV := v;
    end;

begin
end.
```

Der Fehler kann behoben werden, indem die betreffende Anweisung entfernt wird. Eine andere Möglichkeit liegt darin, die Eigenschaft aus dem Automatisierungsabschnitt herauszunehmen.

4.1.3.6.231 Die Direktive '%s' ist im Typ dispinterface nicht erlaubt (E2231)

In einem dispinterface-Typ ist eine unzulässige Klausel enthalten.

```
program Produce;

type
  IBase = dispinterface
  ['{00000000-0000-0000-0000-000000000000}']
    function Get : Integer;

    property BaseValue : Integer read Get;
  end;

  IExt = interface (IBase)
  end;

begin
end.
program Solve;

type
  IBase = dispinterface
  ['{00000000-0000-0000-0000-000000000000}']
    function Get : Integer;

    property BaseValue : Integer;
  end;

begin
end.
```

4.1.3.6.232 Die Klausel '%s' ist im Typ interface nicht erlaubt (E2207)

Die in der Meldung angegebene Klausel darf nicht in einem Schnittstellentyp verwendet werden. Normalerweise deutet dieser Fehler darauf hin, dass eine unzulässige Direktive für ein Eigenschaftsfeld in der Schnittstelle angegeben wurde.

```
program Produce;
type
  Base = interface
    function Reader : Integer;
    procedure Writer(a : Integer);
    property Value : Integer read Reader write Writer stored false;
  end;
begin
end.
```

In diesem Beispiel tritt ein Fehler auf, weil die gespeicherte Direktive nicht in Schnittstellentypen verwendet werden darf.

```
program Solve;
type
  Base = interface
    function Reader : Integer;
    procedure Writer(a : Integer);
    property Value : Integer read Reader write Writer;
  end;
begin
end.
```

Entfernen Sie die betreffende Direktive. Besser wäre es natürlich, das gewünschte Verhalten mit anderen Mitteln zu implementieren.

4.1.3.6.233 Imagebase \$%X ist kein Vielfaches von 64 K. Wird auf \$%X abgerundet (W1043)

Sie können mit der Direktive **\$IMAGEBASE** eine Image-Basisadresse für eine DLL angeben, um sie an einer bestimmten Position in den Speicher zu laden. **\$IMAGEBASE** steuert die Standard-Ladeadresse einer Anwendung, DLL oder Package-Datei. Die als Image-Basis angegebene Adresse muss ein Vielfaches von 64 K sein (d.h. eine Hexadezimalzahl mit vier Nullen am Ende). Andernfalls wird der Wert auf das nächste Vielfache abgerundet, und Sie erhalten diese Fehlermeldung.

4.1.3.6.234 Imagebase ist zu groß - Programmumfang mehr als 2 GB (E2227)

Dieser Fehler kann drei verschiedene Ursachen haben: 1. Die Angabe eines Imagebase-Wertes, der so groß ist, dass der Quelltext für die Anwendung bei der Compilierung mehr als 2GB belegt. 2. Die Angabe eines Imagebase-Wertes in der Kommandozeile, der 2 GB überschreitet. 3. Die Angabe eines Imagebase-Wertes mit der Direktive **\$IMAGEBASE**, der 2 GB überschreitet.

Dieses Problem lässt sich nur durch eine Verringerung der Imagebase-Adresse lösen. Die gesamte Anwendung darf keinesfalls mehr als 2 GB beanspruchen.

4.1.3.6.235 Implements-Klausel kann nicht zusammen mit der Index-Klausel verwendet werden (E2260)

Sie haben versucht, eine index-Klausel zusammen mit der Klausel `implements` zu verwenden. Mithilfe von Indexbezeichnern können mehrere Eigenschaften dieselbe Zugriffsmethode verwenden, auch wenn sie unterschiedliche Werte repräsentieren. Sie können zwar mit `implements` das Implementieren der Schnittstelle einer Eigenschaft der Implementierungsklasse übertragen,

der Indexbezeichner kann aber nicht verwendet werden.

4.1.3.6.236 Der Implements-Getter darf keine dynamische oder message-Methode sein (E2263)

Es wurde versucht, eine dynamic- oder message-Methode zum Zugriff auf eine Eigenschaft zu verwenden, die eine implements-Klausel enthält.

```
program Produce;
type
  I0 = interface
  end;

  T0 = class(TInterfacedObject, I0)
    function getter : I0; dynamic;
    property p0 : I0 read getter implements I0;
  end;

  function T0.getter : I0;
begin
end;

end.
```

Wie in diesem Beispiel gezeigt, ist es ein Fehler, eine dynamic-Methode als Getter für eine Eigenschaft zu deklarieren, die eine implements-Klausel enthält.

```
program Produce;
type
  I0 = interface
  end;

  T0 = class(TInterfacedObject, I0)
    function getter : I0;
    property p0 : I0 read getter implements I0;
  end;

  function T0.getter : I0;
begin
end;

end.
```

Die Lösung besteht darin, die ungültige dynamic- oder method-Deklaration zu entfernen.

4.1.3.6.237 Methoden-Auflösung ist für Interface '%s' nicht erlaubt (E2264)

Es wurde versucht, eine Klausel zum Auflösen einer Methode für eine Schnittstelle zu verwenden, die in einer implements-Klausel angegeben ist.

```
program Produce;
type
  I0 = interface
    procedure i0p0(a : char);
  end;

  T0 = class(TInterfacedObject, I0)
    procedure I0.i0p0 = proc0;
    function getter : I0;
    procedure proc0(a : char);
    property p0 : I0 read getter implements I0;
  end;
```

```
procedure T0.proc0(a : char);
begin
end;
```

```
function T0.getter : I0;
begin
end;
end.
```

In diesem Beispiel wird die Methode proc0 auf die Schnittstellenprozedur i0p0 abgebildet, aber da die Schnittstelle in einer implements-Klausel angegeben wurde, ist die Umbenennung nicht zulässig.

```
program Solve;
type
  I0 = interface
    procedure i0p0(a : char);
  end;

  T0 = class(TInterfacedObject, I0)
    function getter : I0;
    procedure i0p0(a : char);
    property p0 : I0 read getter implements I0;
  end;

  procedure T0.i0p0(a : char);
begin
end;

function T0.getter : I0;
begin
end;
end.
```

Die Lösung besteht darin, die Klausel zur "Namensauflösung" zu entfernen. Dies ist einfach möglich, indem Sie die Prozedur in der Klasse genauso benennen wie die Schnittstellenmethode.

4.1.3.6.238 Implements-Klausel ist nur innerhalb von Klassentypen erlaubt (E2258)

```
program Produce;
type
  IMyInterface = interface
    function getter : IMyInterface;
    property MyInterface: IMyInterface read getter implements IMyInterface;
  end;
end.
```

Bei der Schnittstellendefinition in diesem Beispiel wird versucht, eine implements-Klausel zu verwenden; dies verursacht einen Fehler.

```
program Solve;
type
  IMyInterface = interface
    function getter : IMyInterface;
    property MyInterface: IMyInterface read getter;
  end;
end.
```

Die einzige Lösung besteht darin, die implements-Klausel zu entfernen.

4.1.3.6.239 Implements-Klausel ist nur für Eigenschaften von Klassen- und Interface-Typen erlaubt (E2259)

Es wurde versucht, eine implements-Klausel in eine Eigenschaft eines ungültigen Typs aufzunehmen. Dies ist nur für Klassen- und Schnittstellentypen zulässig.

```
program Produce;
type
  TMyClass = class(TInterfacedObject)
    FInteger : Integer;
    property MyInterface: Integer read FInteger implements Integer;
  end;
end.
```

In diesem Beispiel wird der Fehler dadurch verursacht, dass der Typ Integer zusammen mit einer implements-Klausel verwendet wird.

Die einzige Lösung besteht darin, die implements-Klausel so zu korrigieren, dass sie sich auf einen Klassen- oder Schnittstellentypen bezieht, oder die implements-Klausel ganz zu entfernen.

4.1.3.6.240 Der Implements-Getter muss die Aufrufkonvention %s haben (E2262)

Der Compiler hat einen Getter oder Setter gefunden, der nicht die korrekte Aufrufkonvention hat.

```
program Produce;
type
  I0 = interface
  end;

  T0 = class(TInterfacedObject, I0)
    function getter : I0; cdecl;
    property p0 : I0 read getter implements I0;
  end;

  function T0.getter : I0;
begin
end;
end.
```

In diesem Beispiel verursacht die cdecl für die Funktion getter einen Fehler.

```
program Solve;
type
  I0 = interface
  end;

  T0 = class(TInterfacedObject, I0)
    function getter : I0;
    property p0 : I0 read getter implements I0;
  end;

  function T0.getter : I0;
begin
end;
end.
```

Die einzige Lösung besteht darin, die ungültige Aufrufkonvention von getter zu entfernen.

4.1.3.6.241 **Implements-Klausel ist nur für lesbare Eigenschaften erlaubt (E2261)**

Der Compiler hat eine Nur-Schreiben-Eigenschaft angetroffen, die angibt, eine Schnittstelle zu implementieren. Eine Eigenschaft muss jedoch lesbar sein, damit die implements-Klausel verwendet werden kann.

```
program Produce;
type
  IMyInterface = interface
  end;

  TMyClass = class(TInterfacedObject, IMyInterface)
    FMyInterface: IMyInterface;
    property MyInterface: IMyInterface implements IMyInterface;
  end;
end.
```

Bei der Eigenschaft in diesem Beispiel handelt es sich um eine Nur-Schreiben-Eigenschaft, daher kann sie nicht zur Implementierung einer Schnittstelle verwendet werden.

```
program Solve;
type
  IMyInterface = interface
  end;

  TMyClass = class(TInterfacedObject, IMyInterface)
    FMyInterface: IMyInterface;
    property MyInterface: IMyInterface read FMyInterface implements IMyInterface;
  end;
end.
```

Wenn Sie eine read-Klausel hinzufügen, kann die Eigenschaft die implements-Klausel verwenden.

4.1.3.6.242 **Das Interface '%s' wird in der Liste der Interfaces nicht erwähnt (E2265)**

Eine implements-Klausel verweist auf eine Schnittstelle, die in der Schnittstellenliste der Klasse nicht enthalten ist.

```
program Produce;
type
  IMyInterface = interface
  end;

  TMyClass = class(TInterfacedObject, IUnknown)
    FMyInterface: IMyInterface;
    property MyInterface: IMyInterface read FMyInterface implements IMyInterface;
  end;
end.
```

Im hier gezeigten Beispiel wird die implements-Klausel mit der Schnittstelle IMyInterface verwendet, sie ist jedoch in der Liste der Schnittstellen nicht enthalten.

```
program Solve;
type
  IMyInterface = interface
  end;

  TMyClass = class(TInterfacedObject, IUnknown, IMyInterface)
    FMyInterface: IMyInterface;
    property MyInterface: IMyInterface read FMyInterface implements IMyInterface;
  end;
end.
```

Die hier gezeigte, schnelle Lösung besteht darin, die erforderliche Schnittstelle zur Schnittstellenliste der Klassendefinition hinzuzufügen. Dies erfordert natürlich, dass die Methoden der Schnittstelle implementiert werden.

4.1.3.6.243 Die Unit '%s' wurde implizit in Package '%s' importiert (x1033)

Die angegebene Unit war in der contains-Klausel des Package nicht vorhanden, wird aber von einer Unit importiert, die bereits im Package existiert.

Diese Meldung weist den Programmierer darauf hin, dass er gegen folgende Regel verstoßen hat: Eine Unit darf nicht in mehreren verwandten Packages vorhanden sein.

Das Ignorieren dieser Warnung führt zum Einfügen der Unit in das Package. Wenn Sie die genannte Unit explizit in der contains-Klausel des Package auflisten, erzielen Sie dasselbe Ergebnis und verhindern die Anzeige dieser Warnung. Sie könnten aber auch die Package-Liste ändern, um die genannte Unit aus einem anderen Package zu laden.

```
package Produce;
  contains Classes;
end.
```

In diesem Programmbeispiel verwendet Classes (entweder direkt oder indirekt) consts, TypeInfo und SysUtils. Für jede dieser Units wird eine Warnung ausgegeben.

```
package Solve;
  contains consts, TypeInfo, SysUtils, Classes;
end.
```

Die beste Lösung für dieses Problem besteht darin, explizit alle Units in der contains-Klausel aufzulisten, die in das Package importiert werden (siehe Beispiel).

4.1.3.6.244 Implizite Verwendung der Variants-Unit (W1040)

Wenn Sie in Ihrer Anwendung Variantentypen verwenden, nimmt der Compiler automatisch die Unit Variant in die uses-Klausel auf, weist Sie aber darauf hin, die Datei explizit hinzuzufügen.

4.1.3.6.245 Interface '%s' in '%s' ist nicht vollständig definiert (E2420)

Das Interface ist nicht vollständig definiert. Mit forward deklarierte Interfaces müssen in dem Typabschnitt, in dem sie verwendet werden, deklariert werden. Der folgenden Beispielcode wird nicht compiliert:

```
program Project3;
{$APPTYPE CONSOLE}

type
  TInterface = interface;
  TFoo = class
    type
      TBar = class(TObject, TInterface)
        procedure Bar;
      end;
  end;
  TInterface = interface
    procedure Intf;
  end;

procedure TFoo.TBar.Bar;
begin
```

```
end;
begin
end.
```

4.1.3.6.246 Typ '%s' ist nicht vollständig definiert (E2086)

Dieser Fehler tritt auf, wenn Sie einen Typ referenzieren, der noch nicht vollständig definiert ist, oder wenn Sie in einem type-Abschnitt eine forward-Deklaration für eine Klasse verwenden, ohne später den Typ zu deklarieren.

```
program Produce;
type
  TListEntry = record
    Next: ^TListEntry; (*<-- Hier die Fehlermeldung*)
    Data: Integer;
  end;
  TMyClass = class; (*<-- Hier die Fehlermeldung*)
  TMyClassRef = class of TMyClass;
  TMyClassss = class
    (*...*)
  end;
begin
end.
```

In diesem Beispiel wird ein Record-Typ referenziert, der noch nicht definiert ist. Außerdem ist wegen eines Tippfehlers die Klassendeklaration von TMyClass unvollständig.

```
program Solve;
type
  PListEntry = ^TListEntry;
  TListEntry = record
    Next: PListEntry;
    Data: Integer;
  end;
  TMyClass = class;
  TMyClassRef = class of TMyClass;
  TMyClass = class
    (*...*)
  end;
begin
end.
```

Als Lösung für das erste Problem können Sie eine Typdeklaration für einen Hilfszeiger einführen. Das zweite Problem wird dadurch behoben, dass TMyClass richtig geschrieben wird.

4.1.3.6.247 Anzahl der Elemente (%d) ist von der Deklaration (%d) unterschiedlich (E2072)

Diese Fehlermeldung wird angezeigt, wenn Sie eine typisierte Konstante oder initialisierte Variable eines Array-Typs deklarieren, aber nicht die entsprechende Anzahl von Elementen zuweisen.

```
program Produce;
var
  A: array [1..10] of Integer = (1,2,3,4,5,6,7,8,9);
begin
```

```
end.
```

In diesem Beispiel wird ein Array von 10 Elementen deklariert, in der Initialisierung werden aber nur 9 Elemente angegeben.

```
program Solve;

var
  A: array [1..10] of Integer = (1,2,3,4,5,6,7,8,9,10);

begin
end.
```

Sie brauchen nur das fehlende Element in die Initialisierung aufzunehmen. Bei größeren Arrays kann es schwierig sein, die richtige Elementanzahl zu überprüfen. Sie sollten daher den Quelltext entsprechend strukturieren (z. B. zehn Elemente pro Zeile) oder neben den Elementen einen Kommentar mit einer Nummerierung einfügen.

4.1.3.6.248 Feld '%s' muss initialisiert werden - in CLS-konformen Wertetypen nicht zulässig (E2428)

CLS-konforme Werttypen dürfen keine Felder haben, für die eine Initialisierung erforderlich ist. Siehe ECMA 335, Partition II, Section 12.

4.1.3.6.249 Typ '%s' benötigt Initialisierung - im Variant-Record nicht erlaubt (E2418)

Typ benötigt Initialisierung - im Variant-Record nicht erlaubt. Für Variant-Records sind keine Typen zulässig, die in ihrer Variant-Felderliste initialisiert werden müssen, weil jedes Variant-Feld dieselbe Speicherposition referenziert. Der folgende Beispielcode wird nicht kompiliert, weil der Array-Typ initialisiert werden muss:

```
program Project3;

{$APPTYPE CONSOLE}

type
  TFOO = record
    case Boolean of
      True: (bar: Integer);
      False: (baz: array [0..2] of Integer);
  end;
end.
```

4.1.3.6.250 Lokale Variablen können nicht initialisiert werden (E2195)

Der Compiler verbietet die Verwendung von initialisierten lokalen Variablen.

```
program Produce;

var
  j : Integer;

procedure Show;
  var i: Integer = 151;
begin
end;

begin
end.
```

Die Deklaration und Initialisierung von 'i' in der Prozedur Show ist nicht zulässig.

```

program Solve;

var
  j : Integer;

procedure Show;
  var i: Integer;
begin
  i := 151;
end;

begin
  j := 0;
end.

```

Setzen Sie alle Variablen wie im obigen Beispiel auf bekannte Wert.

4.1.3.6.251 Variablen können nur einzeln initialisiert werden (E2196)

Die Initialisierung von Variablen kann nur durchgeführt werden, wenn die Variablen einzeln deklariert werden.

```

program Produce;

var
  i, j : Integer = 151, 152;

begin
end.

```

Der Compiler verbietet die Deklaration und Initialisierung von mehr als einer Variablen zurzeit.

```

program Solve;

var
  i : Integer = 151;
  j : Integer = 152;

begin
end.

```

Deklarieren Sie einfach jede Variable für sich, damit eine Initialisierung durchgeführt werden kann.

4.1.3.6.252 Thread-lokale Variablen können nicht initialisiert werden (E2194)

Der Compiler lässt eine Initialisierung von Thread-lokalen Variablen nicht zu.

```

program Produce;

threadvar
  tls : Integer = 151;

begin
end.

```

Die nachstehende Deklaration und Initialisierung von 'tls' ist nicht zulässig.

```

program Solve;

threadvar
  tls : Integer;

begin tls := 151;
end.

```

Sie können Thread-lokalen Speicherplatz normal deklarieren und ihn dann im Initialisierungsabschnitt Ihrer Quelldatei

initialisieren.

4.1.3.6.253 Inline-Funktion darf keinen asm-Block haben (E2426)

Inline-Funktionen dürfen keine asm-Blöcke beinhalten. Um diesen Fehler zu vermeiden, entfernen Sie die Direktive **inline** aus Ihrer Funktion oder geben Sie die Anweisungen in dem asm-Block in Pascal-Code an.

4.1.3.6.254 Inline-Direktive im Konstruktor oder Destruktor nicht zulässig (E2442)

Um diesen Fehler zu vermeiden, entfernen Sie die Direktive **inline**.

4.1.3.6.255 Inline-Funktion '%s' wurde nicht expandiert, weil auf Zugriffselement '%s' nicht zugegriffen werden kann (H2444)

Eine Inline-Funktion kann nicht expandiert werden, wenn der Rumpf der Inline-Funktion auf ein **restricted**-Element verweist, das für die Funktion nicht erreichbar ist.

Wenn eine Inline-Funktion beispielsweise auf ein **strict private** Feld verweist und diese Funktion von außerhalb der Klasse aufgerufen wird (z.B. von einer globalen Prozedur), ist das Feld für die Aufrufposition nicht erreichbar und die Inline-Funktion wird nicht expandiert.

4.1.3.6.256 Inline-Methoden dürfen nicht virtual oder dynamic sein (E2425)

Damit eine Inline-Methode beim Compilieren als **inline** eingefügt wird, muss die Methode beim Compilieren gebunden werden. **Virtual** und **dynamic** Methoden werden erst zur Laufzeit gebunden und können daher nicht **inline** eingefügt werden. Wenn Sie Methoden **inline** verwenden möchten, müssen sie **static** sein.

4.1.3.6.257 Verschachtelte Inline-Routine '%s' kann nicht auf Variable '%s' außerhalb des Gültigkeitsbereichs zugreifen (E2449)

Sie können die Direktive **inline** mit verschachtelten Prozeduren und Funktionen verwenden. Eine verschachtelte Prozedur oder Funktion, die auf eine Variable verweist, die für die äußere Prozedur lokal ist, kann jedoch für das Inlining nicht verwendet werden.

4.1.3.6.258 Inline-Funktion '%s' wurde nicht expandiert, weil ihre Unit '%s' nicht in der USES-Anweisung des IMPLEMENTATION-Abschnitts angegeben ist und die aktuelle Funktion eine Inline-Funktion ist (H2445)

Inline-Funktionen werden zwischen zirkulär abhängigen Units nicht expandiert.

4.1.3.6.259 Inline-Funktion '%s' wurde nicht expandiert, weil Unit '%s' in der USES-Liste nicht angegeben ist (H2443)

Diese Situation kann auftreten, wenn eine Inline-Funktion auf einen Typ in einer Unit verweist, die nicht explizit von der Unit der Funktion verwendet wird. Zum Beispiel, wenn die Funktion **inherited** verwendet, um Methoden zu referenzieren, die von einem

entfernten Vorfahren geerbt sind, und die Unit dieses Vorfahren ist nicht explizit in der **uses**-Liste der Unit der Funktion angegeben.

Wenn der Quelltext der Inline-Funktion expandiert werden soll, dann muss die aufrufende Unit die Unit mit dem Vorfahrtyp explizit verwenden.

4.1.3.6.260 Im interface-Abschnitt deklarierte Inline-Funktion darf kein lokales Symbol '%s' verwenden (E2441)

Dieser Fehler tritt auf, wenn eine Inline-Funktion im **interface**-Abschnitt deklariert ist und ein Symbol referenziert, das außerhalb der Unit nicht sichtbar ist. Zum Expandieren der Inline-Funktion in einer anderen Unit müsste auf das lokale Symbol von außerhalb der Unit zugegriffen werden, was nicht zulässig ist.

Um diesen Fehler zu beheben, verlagern Sie die lokale Symboldeklaration in den **interface**-Abschnitt oder wandeln sie in eine Instanzvariable oder eine Klassenvariable des Kalssentyps der Funktion um.

4.1.3.6.261 Konstruktoren können mit Instanzvariablen nicht aufgerufen werden (E2382)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.262 Interner Fehler: %s%d (F2084)

Manchmal wird die Compilierung einer Delphi-Anwendung mit einer Fehlermeldung wie der folgenden abgebrochen:

Interner Fehler: x1234

Diese Meldung bedeutet, dass der Compiler eine Fehlerbedingung entdeckt hat, die nicht auf einen Syntaxfehler zurückgeht, die er jedoch nicht beheben kann.

Die Angabe nach "Interner Fehler" besteht aus einem oder mehr Buchstaben, gefolgt von einer Zahl, die eine compilerinterne Datei- und Zeilennummer darstellt. Obwohl diese Informationen für den Programmierer wenig Aussagekraft haben, helfen Sie uns (Borland) bei der Identifizierung des Problems, wenn Sie uns den Fehler melden. Fügen Sie diese Informationen daher immer Ihrer Fehlerbeschreibung bei.

Siehe auch

Interne Fehler beheben (siehe Seite 1080)

4.1.3.6.263 Interface '%s' besitzt keine Interface-Identifikation (E2232)

Sie haben versucht, eine Schnittstelle, die nicht mit einer GUID definiert wurde, einem GUID-Typ zuzuweisen.

```
program Produce;

type
  IBase = interface
  end;

var
  g : TGUID;

procedure p(x : TGUID);
begin
end;
```

```

begin
  g := IBase;
  p(IBase);
end.

In diesem Beispiel wird der Typ IBase ohne Schnittstelle definiert und kann deshalb nicht einem GUID-Typ zugewiesen werden.

program Solve;

type
  IBase = interface
  ['{00000000-0000-0000-0000-000000000000}']
end;

var
  g : TGUID;

procedure p(x : TGUID);
begin
end;

begin
  g := IBase;
  p(IBase);
end.

```

Dieses Problem lässt sich auf zwei Arten lösen: Entweder vermeiden Sie generell die Zuweisung von Schnittstellen ohne GUID an einen GUID-Typ, oder Sie sorgen bei der Definition der Schnittstelle für die Zuweisung einer GUID. Im vorliegenden Beispiel wird das Problem auf die zweite Art gelöst.

4.1.3.6.264 Implementierung der Interface-Methode %s.%s fehlt (E2291)

Diese Meldung zeigt an, dass Sie vergessen haben, eine Methode zu implementieren, die für ein Interface, das von Ihrem Klassentyp unterstützt wird, erforderlich ist.

4.1.3.6.265 Deklaration von '%s' unterscheidet sich von der Deklaration in interface '%s' (E2211)

Die Methodendeklaration in einer Klasse, die eine Schnittstelle definiert, unterscheidet sich von der Definition, die in der Schnittstelle angegeben ist. Dieser Fehler kann folgende Gründe haben: Ein Parametertyp oder Rückgabewert wurde unterschiedlich deklariert, bei der Methode in der Klasse handelt es sich um eine Botschaftsmethode, der Bezeichner in der Klasse ist ein Feld oder eine Eigenschaft, die nicht mit der Definition in der Schnittstelle übereinstimmt.

```

program Produce;

type
  IBaseIntf = interface
    procedure p0(var x : Shortint);
    procedure p1(var x : Integer);
    procedure p2(var x : Integer);
  end;

  TBaseClass = class (TInterfacedObject)
    procedure p1(var x : Integer); message 151;
  end;

  TExtClass = class (TBaseClass, IBaseIntf)
    p2 : Integer;
    procedure p0(var x : Integer);
    procedure p1(var x : Integer); override;
  end;

```

```
procedure TBaseClass.p1(var x : Integer);
begin
end;
```

```
procedure TExtClass.p0(var x : Integer);
begin
end;
```

```
procedure TExtClass.p1(var x : Integer);
begin
end;
```

```
begin
end.
```

Fehler dieser Art fallen, wie im Beispiel ersichtlich, sofort ins Auge. Die Situation kann aber auch auch komplizierter sein, wie 'p1' zeigt. Da 'p1' eine Prozedur der geerbten Klasse überschreibt, erbt 'p1' auch die in der Basisklasse definierte Virtualität in der Prozedur.

```
program Solve;

type
  IBaseIntf = interface
    procedure p0(var x : Shortint);
    procedure p1(var x : Integer);
    procedure p2(var x : Integer);
  end;

  TBaseClass = class (TInterfacedObject)
    procedure p1(var x : Integer); message 151;
  end;

  TExtClass = class (TBaseClass, IBaseIntf)
    p2 : Integer;

    procedure IBaseIntf.p1 = p3;
    procedure IBaseIntf.p2 = p4;

    procedure p0(var x : Shortint);
    procedure p1(var x : Integer); override;
    procedure p3(var x : Integer);
    procedure p4(var x : Integer);
  end;

  procedure TBaseClass.p1(var x : Integer);
begin
end;

  procedure TExtClass.p0(var x : Shortint);
begin
end;

  procedure TExtClass.p1(var x : Integer);
begin
end;

  procedure TExtClass.p3(var x : Integer);
begin
end;

  procedure TExtClass.p4(var x : Integer);
begin
end;

begin
```

end.

Verwenden Sie (wie im Beispiel gezeigt) für jeden problematischen Bezeichner eine Botschaftsauflösungsklausel. Grundsätzlicher lösen Sie das Problem, wenn Sie vor der Compilierung sicherstellen, dass die Klassenbezeichner mit den Schnittstellenbezeichnern kompatibel sind.

4.1.3.6.266 Interface '%s' bereits implementiert von '%s' (E2208)

Im Vererbungsabschnitt der Klassendefinition wurde die Schnittstelle 'name1' von der Klasse 'name2' mehrmals angegeben.

```
program Produce;
type
  IBaseIntf = interface
  end;

  TBaseClass = class (TInterfacedObject, IBaseIntf, IBaseIntf)
  end;

begin
end.
```

In diesem Beispiel wird die Schnittstelle IBaseIntf interface im Vererbungsabschnitt der Definition von TBaseClass mehrmals angegeben. Eine Klasse darf aber dieselbe Schnittstelle nicht mehrmals implementieren. Wenn das der Fall ist, gibt der Compiler diese Fehlermeldung aus.

```
program Solve;
type
  IBaseIntf = interface
  end;

  TBaseClass = class (TInterfacedObject, IBaseIntf)
  end;

begin
end.
```

Für diesen Fehler gibt es nur eine Lösung: Sie müssen sicherstellen, dass die Schnittstelle nur einmal im Vererbungsabschnitt der Klassendefinition angegeben ist.

4.1.3.6.267 Integerkonstante zu lang (E2102)

Sie haben eine Integer-Konstante angegeben, deren Darstellung mehr als 64 Bit erfordert.

```
program Produce;
const
  VeryBigHex = $8000000000000001;

begin
end.
```

Die Konstante in diesem Beispiel kann nicht in 64 Bit dargestellt werden.

```
program Solve;
const
  BigHex = $8000000000000001;

begin
end.
```

Überprüfen Sie Ihre Konstanten, und stellen Sie sicher, dass sie in 64 Bit dargestellt werden können.

4.1.3.6.268 Ungültige Typumwandlung (E2089)

Diese Fehlermeldung weist auf eine Typumwandlung hin, die nicht den Regeln entspricht. Erlaubt sind folgende Umwandlungen:

- Ordinaler oder Zeigertyp in einen anderen ordinalen oder Zeigertyp
- Zeichen, String, Zeichen- oder pChar-Array in einen String
- Ordinaler Typ, Real, String oder Variante in eine Variante
- Variante in einen ordinalen Typ, Real, String oder eine Variante
- Variablenreferenz in einen beliebigen Typ derselben Größe

Umwandlungen von Real- in Integer-Typen können mit den Standardfunktionen Trunc und Round durchgeführt werden.

Weitere Umwandlungsfunktionen sind z. B. Ord und Chr.

```
program Produce;

begin
  Writeln( Integer(Pi) );
end.
```

Es sollte wie in C eine Gleitkommakonstante in einen Integer-Wert konvertiert werden.

```
program Solve;

begin
  Writeln( Trunc(Pi) );
end.
```

In Delphi gibt es für diesen Zweck eigene Umwandlungsfunktionen.

4.1.3.6.269 Codepage '%s' ist auf diesem Rechner nicht installiert (E2424)

Diese Meldung tritt auf, wenn Sie eine Codepage mit der Befehlszeilenoption `--codepage=nnn` angeben und die Codepage auf dem Rechner nicht zur Verfügung steht.

In der Dokumentation zu Ihrem Betriebssystem finden Sie Einzelheiten zum Installieren von Codepages.

4.1.3.6.270 Fehlendes oder ungültiges bedingtes Symbol in der Anweisung '\$%s' (E2173)

Nach den Anweisungen \$IFDEF, \$IFNDEF, \$DEFINE and \$UNDEF muss ein Symbol stehen.

```
program Produce;

(*$IFDEF*)
(*$ENDIF*)
```

```
begin
end.
```

Die bedingte Anweisung \$IFDEF wurde hier nicht ordnungsgemäß festgelegt, so dass ein Fehler ausgelöst wird.

```
program Solve;

(*$IFDEF WIN32*)
(*$ENDIF*)

begin
```

end.

Die Lösung dieses Problems liegt darin, darauf zu achten, dass ein zu testendes Symbol nach der zugehörigen Anweisung steht.

4.1.3.6.271 Ungültige Compiler-Direktive: '%s' (x1030)

Diese Meldung wird angezeigt, wenn eine Compiler-Direktive oder Befehlszeilenoption einen Fehler enthält. Die folgende Aufstellung zeigt einige der möglichen Ursachen:

- Eine externe Deklaration enthält Syntaxfehler.
- Eine Befehlszeilenoption oder eine Option in der Datei DCC32.CFG ist fehlerhaft oder wurde vom Compiler nicht erkannt. So ist beispielsweise die Option "-\$M100" ungültig, da die minimale Stack-Größe mindestens 1024 sein muss.
- Der Compiler hat eine \$XXXXX-Direktive gefunden, aber nicht erkannt. Sie enthält wahrscheinlich einen Syntaxfehler.
- Der Compiler hat die Direktive \$ELSE oder \$ENDIF, aber kein vorhergehendes \$IFDEF, \$IFNDEF oder \$IFOPT gefunden.
- Nach (*\$IFOPT*) wurde kein Schalter mit dem Zeichen + oder - eingegeben.
- Nach der langen Form einer Direktive fehlt ON oder OFF.
- Auf eine Direktive mit einem numerischen Parameter folgt keine gültige Zahl.
- Nach der Direktive \$DESCRIPTION wurde kein String eingegeben.
- Nach der Direktive \$APPTYPE wurde nicht CONSOLE oder GUI eingegeben.
- Nach der Direktive \$ENUMSIZE (Kurzform \$Z) wurde nicht 1, 2 oder 4 eingegeben.

```
(*$Description Copyright CodeGear 2007*)      (*<-- Hier der Fehler*)
program Produce;                           (*<-- Hier der Fehler*)
(*$AppType Console*)                      (*<-- Hier der Fehler*)

begin
(*$If O+*)                                (*<-- Hier der Fehler*)
Writeln('Optimizations are ON');
(*$Else*)                                 (*<-- Hier der Fehler*)
Writeln('Optimizations are OFF');
(*$Endif*)                                (*<-- Hier der Fehler*)
Writeln('Hello world!');
end.
```

Das Beispiel zeigt drei typische Fehlersituationen. Die letzten beiden Fehler wurden durch den Compiler verursacht, der die Direktive \$If nicht erkannt hat.

```
(*$Description 'Copyright CodeGear 2007'* ) (*String erforderlich*)
program Solve;                           (*AppType*)
(*$AppType Console*)                     (*OK*)

begin
(*$IfOpt O+*)                            (*IfOpt*)
  Writeln('Optimizations are ON');
(*$Else*)                                (*OK*)
  Writeln('Optimizations are OFF');
(*$Endif*)                                (*OK*)
  Writeln('Hello world!');
end.
```

\$Description benötigt einen in Anführungszeichen gesetzten String, \$AppType muss richtig geschrieben werden, und die Testoptionen werden mit \$IfOpt durchgeführt. Mit diesen Änderungen kann das Beispiel erfolgreich kompiliert werden.

4.1.3.6.272 read/write für CLR-Ereignisse nicht zulässig. Verwenden Sie eine Include/Exclude-Prozedur (E2298)

Multicast-Ereignisse dürfen nicht zugewiesen oder gelesen werden wie traditionelle Delphi-read/write-Ereignisse.

Verwenden Sie Include/Exclude, um Methoden hinzuzufügen oder zu entfernen.

4.1.3.6.273 Ungültige Botschaftsparameterliste (E2138)

Eine Botschaftsprozedur kann nur einen Parameter haben (var). Der Typ des Parameters wird nicht geprüft.

```
program Produce;

type
  Base = class
    procedure Msg1(x : Integer); message 151;
    procedure Msg2(VAR x, y : Integer); message 152;
  end;

  procedure Base.Msg1(x : Integer);
begin
end;

  procedure Base.Msg2(VAR x, y : Integer);
begin
end;

begin
end.
```

Im ersten Fall besteht der Fehler darin, dass es sich nicht um einen var-Parameter handelt. Im zweiten Fall ist mehr als ein Parameter deklariert.

```
program Solve;

type
  Base = class
    procedure Msg1(VAR x : Integer); message 151;
    procedure Msg2(VAR y : Integer); message 152;
  end;

  procedure Base.Msg1(VAR x : Integer);
begin
end;

  procedure Base.Msg2(VAR y : Integer);
begin
end;

begin
end.
```

Die Lösung besteht in beiden Fällen darin, in der Deklaration der Botschaftsmethode nur einen Parameter (var) anzugeben.

4.1.3.6.274 Ein in einer unterstützenden Klasse eingeführter Konstruktor muss den parameterlosen Konstruktor der unterstützten Klasse als erste Anweisung aufrufen. (E2296)

Die erste Anweisung in dem Konstruktor einer unterstützenden Klasse muss "inherited Create;" lauten.

4.1.3.6.275 Eine von '%s' abgeleitete, unterstützende Klasse kann nur Klassen unterstützen, die von '%s' abstammen. (E2294)

Der in der "for"-Klausel einer Deklaration für eine unterstützende Klasse angegebene Objekttyp ist kein Nachkomme des Objekttyps, der in der "for"-Klausel des Vorfahrtyps der unterstützenden Klasse festgelegt wurde.

4.1.3.6.276 Der Name des Schlüsselcontainers '%s' ist nicht vorhanden (E2387)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.277 Unerkannte Schlüsseldatei '%s' für starke Namen (E2388)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.278 %s kann nicht auf ein rechteckiges, dynamisches Array angewendet werden (E2432)

Dieser Fehler kann auftreten, wenn Sie versuchen, ein dynamisch zugewiesenes, rechteckiges Array an die Funktion Low oder High zu übergeben.

Wenn Sie diesen Fehler erhalten, verwenden Sie ein statisches oder nicht rechteckiges Array. Einzelheiten dazu finden Sie in der Delphi-Sprachreferenz.

Siehe auch

Strukturierte Typen (siehe Seite 902)

4.1.3.6.279 Ungültige Operatordeklaration (E2393)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.280 '%s' wurde vorher nicht als PROPERTY deklariert (E2174)

Sie haben versucht, eine Eigenschaft durch Redeklaration auf eine andere, höhere Sichtbarkeitsebene zu stellen, aber <Element> in der Basis-Klasse wurde nicht als Eigenschaft deklariert. -W

```
program Produce;
(*$WARNINGS ON*)

type
  Base = class
  protected
    Caption : String;
    Title : String;
    property TitleProp : string read Title write Title;
  end;

  Derived = class (Base)
  public
    property Title read Caption write Caption;
  end;
```

begin
end.

Der Zweck einer Neudeklaration von 'Derived.Title' ist es, das Feld zu verändern, das zum Lesen und Schreiben der Eigenschaft 'Title' benutzt wird, und es der höheren public-Sichtbarkeit zur Verfügung zu stellen. Der Programmierer wollte eigentlich TitleProp benutzen und nicht Title.

```
program Solve;  
(*$WARNINGS ON*)  
  
type  
  Base = class  
  protected  
    Caption : String;  
    Title : String;  
    property TitleProp : string read Title write Title;  
  end;  
  
  Derived = class (Base)  
  public  
    property TitleProp read Caption write Caption;  
    property Title: string read Caption write Caption;  
  end;  
  
begin  
end.
```

Dieser Fehler kann auf mehrere Arten beseitigt werden. Die erste Möglichkeit, die wahrscheinlich am häufigsten genutzt wird, ist die Festlegung der korrekten zu redeklarierenden Eigenschaft. Die zweite Möglichkeit, die in der Redeklaration von Title deutlich wird, beseitigt das Problem durch explizite Erstellung einer neuen Eigenschaft mit derselben Bezeichnung wie ein Feld in der Basisklasse. Diese neue Eigenschaft verbirgt das Basisfeld, auf das daher ohne Typumwandlung kein Zugriff mehr möglich ist. (Hinweis: Wenn die Warnfunktion aktiviert ist, wird mit der Neudeklaration von Title eine Warnung ausgegeben, mit der Sie davon in Kenntnis gesetzt werden, dass das Mitglied der Basisklasse durch die Neudeklaration verborgen wird.)

4.1.3.6.281 STATIC kann nur für nicht-virtuelle Klassenmethoden verwendet werden (E2376)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.282 Assemblierung '%s' konnte nicht importiert werden, weil sie den Namespace '%s' enthält (E2415)

Die Unit Borland.Delphi.System darf nur von der Assemblierung Borland.Delphi.dll geladen werden. Dieser Fehler tritt auf, wenn eine alternative Assemblierung versucht, Borland.Delphi.System laden.

4.1.3.6.283 Package '%s' konnte nicht importiert werden, weil es die System-Unit '%s' enthält (E2416)

Die Unit Borland.Delphi.System darf nur vom Package Borland.Delphi.dll geladen werden. Dieser Fehler tritt auf, wenn ein alternatives Package versucht, Borland.Delphi.System laden.

4.1.3.6.284 UCS-4-Textcodierung nicht unterstützt. Konvertieren Sie in UCS-2 oder UTF-8 (F2438)

Dieser Fehler tritt auf, wenn eine Quelldatei in UCS-4 codiert ist, was im BOM (Byte-Order-Mark) angegeben ist. Der Compiler unterstützt das Compilieren von Quelldateien mit der Codierung UCS-4 Unicode nicht. Um dieses Problem zu beheben, konvertieren Sie die Quelldatei nach UCS-2 oder UTF-8.

4.1.3.6.285 Ungültiger Versions-String '%s' in %s angegeben (E2386)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.286 LOOP/JCXZ außerhalb des Wertebereichs (E2120)

Sie haben ein LOOP- oder JCXZ-Ziel außerhalb des gültigen Bereichs angegeben. Der Fehler sollte nicht auftreten, da der gültige Bereich für LOOP- und JCXZ-Anweisungen 2 GB beträgt.

4.1.3.6.287 Labeldeklaration ist im Interface-Abschnitt nicht erlaubt (E2049)

Dieser Fehler tritt auf, wenn Sie eine Sprungmarke im interface-Abschnitt einer Unit deklarieren.

```
unit Produce;
interface
label 99;
implementation
begin
99:
end.
```

Sprungmarken dürfen nicht im interface-Abschnitt einer Unit deklariert werden.

```
unit Solve;
interface
implementation
label 99;
begin
99:
end.
```

Sie müssen die Deklaration im implementation-Abschnitt durchführen.

4.1.3.6.288 Label ist bereits definiert: '%s' (E2073)

Diese Fehlermeldung wird angezeigt, wenn eine Sprungmarke mehrfach angegeben wird.

```
program Produce;
label 1;
begin
1:
  goto 1;
1:      (*<-- Hier die Fehlermeldung*)
end.
```

In diesem Beispiel wird die Sprungmarke 1 zweimal verwendet.

```
program Solve;
label 1;
begin
```

```
1:  
  goto 1;  
end.
```

Vergewissern Sie sich, dass jede Sprungmarke nur einmal vorhanden ist.

4.1.3.6.289 Label wurde deklariert und referenziert, aber nicht gesetzt: "%s" (E2074)

Sie haben eine Sprungmarke in Ihrem Programm deklariert und verwendet, aber ihre Definition kann im Quelltext nicht gefunden werden.

```
program Produce;  
  
procedure Labeled;  
label 10;  
begin  
  goto 10;  
end;  
  
begin  
end.
```

label 10 wird deklariert und in der Prozedur Labeled verwendet, aber die Label-Definition fehlt.

```
program Produce;  
  
procedure Labeled;  
label 10;  
begin  
  goto 10;  
  10:  
end;  
  
begin  
end.
```

Achten Sie darauf, dass für jede Sprungmarke, die Sie deklariert und verwendet haben, im selben Gültigkeitsbereich eine Definition vorhanden ist.

4.1.3.6.290 Zeile zu lang (mehr als 1023 Zeichen) (F2069)

Diese Fehlermeldung wird angezeigt, wenn eine Zeile in der Quelltextdatei mehr als 1023 Zeichen enthält.

Normalerweise können lange Zeilen in mehrere kürzere Zeilen aufgeteilt werden.

Wenn Sie einen sehr langen literalen String benötigen, können Sie diesen in mehreren aufeinander folgenden Zeilen in einzelne Strings aufteilen und diese mit dem Operator "+" verketten.

4.1.3.6.291 Assemblierungsübergreifende protected-Referenz auf [%s]%-s.%s in %s.%s (E2364)

In Delphi für .NET kann auf Elemente mit der Sichtbarkeit **protected** nicht außerhalb der Assemblierung, in der sie definiert sind, zugegriffen werden. Falls möglich sollten Sie die als **public** bereitgestellten Elemente der Klasse einsetzen.

Andere Möglichkeiten, diesen Fehler zu umgehen, sind:

- Erweitern Sie die Sichtbarkeit der Elemente von **protected** zu **public**, damit auf sie außerhalb ihrer Assemblierung zugegriffen werden kann.

- Binden Sie die Assemblierung ein, in der das **protected**-Element definiert ist, damit diese Assemblierung in die zu erstellende Assemblierung einbezogen ist. Der Zugriff erfolgt dann innerhalb der Assemblierung.

Siehe auch

Klassen und Objekte (siehe Seite 948)

Delphi-Units in eine Anwendung linken (siehe Seite 10)

4.1.3.6.292 Fehler beim Konvertieren des Strings '%s' in Unicode. String wurde abgeschnitten. Ist die Umgebungsvariable LANG korrekt gesetzt? (W1042)

Diese Meldung wird angezeigt, wenn Sie Strings in das Unicode-Format konvertieren und der String Zeichen enthält, die im aktuellen Gebietsschema nicht zulässig sind. Dies ist beispielsweise beim Konvertieren von WideString in AnsiString oder beim Versuch, japanische Zeichen in einem englischen Gebietsschema anzuzeigen, der Fall.

4.1.3.6.293 Lokaler PInvoke-Quelltext wurde nicht erstellt, weil die externe Routine '%s' im Package '%s' mit package-lokalen Typen in den benutzerdefinierten Attributen definiert wurde (W1053)

Diese Warnung kann ausgelöst werden, wenn ein externes Package mit PInvoke auf Win32-Bibliotheksquelltext zugreift, und dieses Package die PInvoke-Definition über einen **public**-Export bereitstellt. In diesen Fällen versucht der Compiler, eine direkte Verknüpfung zu der Win32-Bibliothek durch Kopieren der PInvoke-Definition in die lokale Assemblierung herzustellen, anstatt mit dem **public**-Export in dem externen Package zu verknüpfen. Dies ist sicherer und kann auch die Laufzeitperformanz verbessern.

Diese Warnmeldung wird ausgegeben, wenn der Compiler die PInvoke-Definition nicht lokal erstellen kann, weil die externe Assemblierung lokal definierte Typen für benutzerdefinierte Attribute verwendet. Um diese Warnung zu vermeiden, müssen Sie das Package so ändern, dass keine lokal definierten Typen für benutzerdefinierte Attribute in exportierten Funktionen oder Prozeduren verwendet werden.

Beispielsweise stellt im folgenden Quelltext die Unit ExternalPackagedUnit die externe Funktion Beep in Kernel32 über die Funktion TFoo.Beep mit dem benutzerdefinierten Attribut MyAttribute bereit:

```
unit ExternalPackagedUnit;

interface

function Beep(dwFreq, dwDuration: MyLongWord): Boolean; static; stdcall;

implementation

type
  MyAttribute = class(System.Attribute)
  .
  .
  .
end;

[MyAttribute]
function Beep(dwFreq, dwDuration: LongWord): Boolean; stdcall; external 'kernel32' name 'Beep';

end.
```

Wenn ein Programm, das ExternalPackagedUnit verwendet, compiliert werden soll, kann der Compiler keine direkte Verknüpfung zu der Funktion Beep in Kernel32 herstellen, weil der Typ MyAttribute in ExternalPackagedUnit lokal definiert ist, und diese Warnung wird ausgegeben. In dem Beispiel kann dieses Problem behoben werden, indem der Typ MyAttribute als **public** definiert wird (durch Verlagern der Deklaration aus dem **implementation**-Abschnitt in den

interface-Abschnitt), oder indem das benutzerdefinierte Attribut aus der Funktion Beep entfernt wird.

Siehe auch

Platform Invoke mit RAD Studio

4.1.3.6.294 Lokale Prozedur/Funktion '%s' wurde Prozedurenvariable zugewiesen (E2094)

Diese Fehlermeldung wird angezeigt, wenn Sie versuchen, eine lokale Prozedur einer Prozedurenvariable zuzuweisen oder sie als prozeduralen Parameter zu übergeben.

Dies ist nicht erlaubt, da die lokale Prozedur dann auch aufgerufen werden könnte, wenn die umgebende Prozedur nicht aktiv ist. Wenn die lokale Prozedur in diesem Fall versucht, auf Variablen der umgebenden Prozedur zuzugreifen, hat dies einen Programmabsturz zur Folge.

```
program Produce;

var
  P: Procedure;

procedure Outer;

  procedure Local;
  begin
    Writeln('Local is executing');
  end;

begin
  P := Local;          (*<-- Hier die Fehlermeldung*)
end;

begin
  Outer;
  P;
end.
```

Hier wird versucht, einer Prozedurenvariable eine lokale Prozedur zuzuweisen. Dies ist nicht erlaubt (weil zur Laufzeit unsicher).

```
program Solve;

var
  P: Procedure;

procedure NonLocal;
begin
  Writeln('NonLocal is executing');
end;

procedure Outer;

begin
  P := NonLocal;
end;

begin
  Outer;
  P;
end.
```

Verlegen Sie die lokale Prozedur aus der umgebenden Prozedur heraus.

4.1.3.6.295 Thread-lokale Variablen können nicht lokal zu einer Funktion sein (E2189)

Thread-lokale Variablen müssen mit globalem Gültigkeitsbereich deklariert werden.

```
program Produce;

procedure NoTLS;
  threadvar
    x: Integer;
begin
end;

begin
end.
```

Eine Thread-Variable kann nicht als lokal zu einer Prozedur deklariert werden.

```
program Solve;

threadvar
  x: Integer;

procedure YesTLS;
  var
    localX : Integer;
begin
end;

begin
end.
```

Zur Vermeidung dieses Fehlers gibt es zwei einfache Möglichkeiten. Zum einen kann der Abschnitt `threadvar` auf lokalen Gültigkeitsbereich gebracht werden. Zum anderen kann der Abschnitt `threadvar` in der Prozedur in einen normalen `var`-Abschnitt geändert werden. Beachten Sie, dass für den Fall, dass die Compiler-Hinweise aktiviert sind, ein Hinweis darüber gegeben wird, dass `localX` zwar deklariert, aber niemals verwendet wird.

4.1.3.6.296 Inline-Methodensichtbarkeit muss niedriger oder gleich der Sichtbarkeit des Zugriffselements '%s.%s' sein (H2440)

Ein Element, auf das im Rumpf einer Inline-Funktion zugegriffen wird, muss von jeder Position aus erreichbar sein, von der die Inline-Methode aufgerufen wird. Daher muss das Element mindestens dieselbe Sichtbarkeit haben wie die Inline-Methode.

Hier ein Beispiel für Quelltext, der zu diesem Fehler führt:

```
type
  TFOO = class
  private
    PrivateMember: Integer;
  public
    function PublicFunc: Integer; inline;
  end;

function TFOO.PublicFunc: Integer;
begin
  Result := Self.PrivateMember;
end;
```

Weil `Result := Self.PrivateMember;` überall dort eingefügt wird, wo `PublicFunc` aufgerufen wird, muss `PrivateMember` von jeder dieser Positionen aus erreichbar sein.

Um diesen Fehler zu beheben, entfernen Sie die Direktive **inline** oder passen die Sichtbarkeit der Inline-Methode oder des Elements, auf das sie zugreift, an.

4.1.3.6.297 Unterer Bereich überschreitet oberen Bereich (E2011)

Diese Fehlermeldung wird angezeigt, wenn die Untergrenze eines Subrange-Typs über dessen Obergrenze liegt oder wenn die Untergrenze eines Case-Label-Bereichs über dessen Obergrenze liegt.

```
program Produce;
type
  SubrangeType = 1..0;           (*Fehlermeldung: Unterer Bereich überschreitet oberen
Bereich *)
begin
  case True of
    True..False:           (*Fehlermeldung: Unterer Bereich überschreitet oberen
Bereich *)
      Writeln('Expected result');
    else
      Writeln('Unexpected result');
  end;
end.
```

Im obigen Beispiel löst der Compiler eine Fehlermeldung aus, anstatt die Bereiche als leer zu behandeln. Aller Wahrscheinlichkeit nach war das Vertauschen der Grenzen nicht beabsichtigt.

```
program Solve;
type
  SubrangeType = 0..1;
begin
  case True of
    False..True:
      Writeln('Expected result');
    else
      Writeln('Unexpected result');
  end;
end.
```

Stellen Sie sicher, dass Sie die Grenzen in der richtigen Reihenfolge festgelegt haben.

4.1.3.6.298 Falsche GUID-Syntax (E2204)

Die im Programmtext vorhandene GUID ist fehlerhaft. Eine GUID muss folgendes Format haben: 00000000-0000-0000-0000-000000000000.

4.1.3.6.299 Zu wenig Arbeitsspeicher (F2046)

Der verfügbare Arbeitsspeicher reicht nicht aus.

Dieser Fall sollte nur selten eintreten. Vergewissern Sie sich, dass die Auslagerungsdatei groß genug ist und ausreichend freier Festplattenspeicher zur Verfügung steht.

4.1.3.6.300 Linker-Fehler: %s (x1054)

Diese Meldung zeigt eine Warnung oder anderen Text an, der durch die Direktive \$MESSAGE generiert wird.

4.1.3.6.301 Methodendeklarationen nicht zulässig in anonymen Record- oder lokalen Record-Typ (E2433)

Record-Typen, die in lokalen Gültigkeitsbereichen oder in Variablen Deklarationen deklariert sind, dürfen nur Feld Deklarationen enthalten. Für erweiterte Features in Record-Typen (wie z.B. Methoden, Eigenschaften und verschachtelte Typen) muss der Record-Typ explizit als globaler Typ definiert werden.

4.1.3.6.302 Methodenbezeichner erwartet (E2096)

Dieser Fehler kann in verschiedenen Situationen auftreten:

- Eigenschaften in einem automated-Abschnitt müssen Zugriffsmethoden haben. Sie dürfen in ihren read- oder write-Klauseln keine Felder verwenden.
- Sie versuchen, eine Klassenmethode mit der Syntax "Klassentyp.Methodename" aufzurufen, aber "Methodename" ist kein Methodename.
- Sie versuchen, eine geerbte Methode mit der Syntax "Inherited Methodename" aufzurufen, aber "Methodename" ist kein Methodename.

```
program Produce;

type
  TMyBase = class
    Field: Integer;
  end;
  TMyDerived = class (TMyBase)
    Field: Integer;
    function Get : Integer;
  Automated
    property Prop: Integer read Field;      (*<-- Hier die Fehlermeldung*)
  end;

  function TMyDerived.Get: Integer;
begin
  Result := TMyBase.Field;                  (*<-- Hier die Fehlermeldung*)
end;

begin
end.
```

In diesem Beispiel wird eine automated-Eigenschaft deklariert, die direkt auf ein Feld zugreift. Der zweite Fehler wird durch den Zugriff auf ein Feld der Basisklasse verursacht – auch dies ist nicht erlaubt.

```
program Solve;

type
  TMyBase = class
    Field: Integer;
  end;
  TMyDerived = class (TMyBase)
    Field: Integer;
    function Get : Integer;
  Automated
    property Prop: Integer read Get;
  end;

  function TMyDerived.Get: Integer;
begin
  Result := TMyBase(Self).Field;
end;
```

```
begin
  Writeln( TMyDerived.Create.Prop );
end.
```

Das erste Problem wird dadurch behoben, dass für den Zugriff auf das Feld eine Methode verwendet wird. Das zweite Problem wird behoben, indem der Zeiger Self in den Typ der Basisklasse konvertiert wird und erst dann der Zugriff auf das Feld erfolgt.

4.1.3.6.303 Metadaten - Falsche Eingabeparameter (E2348)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.304 Metadaten - Falsche binäre Signatur (E2347)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.305 Metadaten - typeref kann nicht aufgelöst werden (E2349)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.306 Metadaten - Versuch, ein Objekt zu definieren, das bereits vorhanden ist (E2345)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.307 Metadaten - Eine erforderliche GUID wurde nicht bereitgestellt (E2346)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.308 Metadaten - Kein logischer Platz zum Erzeugen weiterer Benutzer-Strings vorhanden (E2350)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.309 Getter oder Setter für Eigenschaft '%s' kann nicht gefunden werden (E2234)

Während der Übersetzung einer Unit in eine C++ Header-Datei konnte der Compiler ein benanntes Symbol nicht finden, das zum Lesen oder Setzen einer Eigenschaft benötigt wird. Die Ursache hierfür ist meist eine Verschachtelung von Records, wobei der Zugriff durch ein Feld im verschachtelten Record erfolgt.

4.1.3.6.310 ENDIF fehlt (E2095)

Diese Fehlermeldung wird ausgegeben, wenn der Compiler nach einer \$IFDEF-, \$IFNDEF- oder \$IFOPT-Direktive keine entsprechende \$ENDIF-Direktive findet.

```
program Produce;
(*$APPTYPE CONSOLE*)
```

```

begin
(*$IfOpt O+*)
  Writeln('Compiled with optimizations');
(*$Else*)
  Writeln('Compiled without optimizations');
(*$Endif*)
end.                                     (*<-- Hier die Fehlermeldung*)

```

In diesem Beispiel fehlt das Zeichen \$ in der (*\$Endif*)-Direktive. Der Compiler hält sie deshalb für einen Kommentar.

```

program Solve;
(*$APPTYPE CONSOLE*)
begin
(*$IfOpt O+*)
  Writeln('Compiled with optimizations');
(*$Else*)
  Writeln('Compiled without optimizations');
(*$Endif*)
end.

```

Stellen Sie sicher, dass alle bedingten Direktiven eine gültige \$ENDIF-Direktive haben.

4.1.3.6.311 Zugriffsmethode zum Hinzufügen oder Entfernen für Ereignis '%s' nicht gefunden (E2403)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.312 Der Vorfahrtyp '%s' besitzt keinen zugreifbaren Standardkonstruktor (E2253)

Für den Vorfahr der zu compilierenden Klasse gibt es keinen zugreifbaren Standardkonstruktor. Dieser Fehler tritt nur mit der Byte-Code-Version des Compilers auf.

4.1.3.6.313 Operator oder Semikolon fehlt (E2066)

Diese Fehlermeldung tritt auf, wenn zwischen zwei Teilausdrücken kein Operator oder zwischen zwei Anweisungen kein Semikolon eingegeben wird.

Das Semikolon fehlt oft in der vorhergehenden Zeile.

```

program Produce;
var
  I: Integer;
begin
  I := 1 2                         (*<-- Hier die Fehlermeldung*)
  if I = 3 then                      (*<-- Hier die Fehlermeldung*)
    Writeln('Fine')
end.

```

Die erste Anweisung in diesem Beispiel enthält zwei Fehler: es fehlen der Operator "+" und das Semikolon. Der erste Fehler wird in dieser Zeile, der zweite in der folgenden ausgegeben.

```

program Solve;
var
  I: Integer;
begin
  I := 1 + 2;                      (*Es fehlen der Operator '+' und ein Semikolon*)
  if I = 3 then
    Writeln('Fine')
end.

```

Vergewissern Sie sich, dass Sie die benötigten Operatoren und Semikolons eingegeben haben.

4.1.3.6.314 Package '%s' wird benötigt, konnte aber nicht gefunden werden (E2202)

Das in der Meldung angegebene Package ist zwar in der Package-Liste aufgeführt (entweder explizit oder indirekt über die requires-Klausel einer anderen, in der Package-Liste enthaltenen Unit), kann aber vom Compiler nicht gefunden werden.

Stellen Sie sicher, dass sich die DCP-Datei für das Package unter den im Bibliothekspfad aufgeführten Units befindet.

4

4.1.3.6.315 Nicht genügend Parameter (E2035)

Diese Fehlermeldung tritt auf, wenn beim Aufruf einer Prozedur oder einer Funktion weniger Parameter angegeben werden als in der Deklaration der Prozedur oder Funktion festgelegt wurden.

Dies kann auch bei Aufrufen von Standard-Prozeduren oder -Funktionen der Fall sein.

```
program Produce;
var
  X: Real;
begin
  Val('3.141592', X); (*<-- Hier die Fehlermeldung*)
end.
```

Die Standard-Prozedur Val hat einen zusätzlichen Parameter, mit dem ein Fehlercode zurückgegeben werden kann. In diesem Beispiel wurde dieser Parameter nicht angegeben.

```
program Solve;
var
  X: Real;
  Code: Integer;
begin
  Val('3.141592', X, Code);
end.
```

Als typische Reaktion auf diesen Fehler werden Sie den Aufruf mit der Deklaration der aufgerufenen Prozedur oder mit der Hilfefunktion vergleichen und feststellen, dass Sie einen Parameter vergessen haben, der angegeben werden muss.

4.1.3.6.316 Fehlender Parametertyp (E2067)

Diese Fehlermeldung wird angezeigt, wenn in einer Parameterliste der Typ eines Wertparameters nicht angegeben ist.

Konstanten- und Variablenparameter können ohne Typ übergeben werden.

```
program Produce;

procedure P(I;J: Integer); (*<-- Hier die Fehlermeldung*)
begin
end;

function ComputeHash(Buffer; Size: Integer): Integer; (*<-- Hier die Fehlermeldung*)
begin
end;

begin
end.
```

Der Prozedur P sollen eigentlich zwei Integer-Parameter übergeben werden. Nach dem ersten Argument steht aber ein Semikolon anstelle eines Kommas. Der erste Parameter der Funktion ComputeHash ist nicht typisiert. Dies ist aber nur bei

Variablen- oder Konstantenparametern möglich, nicht bei Wertparametern.

```
program Solve;

procedure P(I,J: Integer);
begin
end;

function ComputeHash(const Buffer; Size: Integer): Integer;
begin
end;

begin
end.
```

Die Lösung besteht darin, zwischen den Parametern von P ein Komma einzugeben und den Parameter Buffer von ComputeHash als Konstantenparameter zu deklarieren.

4.1.3.6.317 RLINK32.DLL kann nicht geladen werden (E2151)

Die Datei RLINK32.DLL konnte nicht gefunden werden. Achten Sie darauf, dass sie sich im angegebenen Verzeichnis befindet.

Wenden Sie sich an CodeGear, wenn Sie diese Fehlermeldung erhalten.

4.1.3.6.318 READ/WRITE-Eigenschaftszugriffsmethoden dürfen nicht mit ADD/REMOVE-Zugriffsmethoden gemischt werden (E2404)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.319 Mehrere Klassenkonstruktoren in Klasse %s: %s und %s (E2359)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.320 '%s' kann nicht mehrfach exportiert werden (E2287)

Diese Meldung wird in diesem Produkt nicht verwendet.

4.1.3.6.321 Unit-Namen stimmen nicht überein: '%s' '%s' (E2085)

Da bei dem Unit-Namen in der obersten Unit zwischen Groß- und Kleinschreibung unterschieden wird, muss die Schreibweise genau übereinstimmen. Beim Unit-Namen findet diese Unterscheidung nur in der Unit-Deklaration statt.

4.1.3.6.322 Array-Typ erforderlich (E2016)

Diese Fehlermeldung wird angezeigt, wenn Sie versuchen, einen Index in einen Operanden zu setzen, der kein Array ist, oder wenn Sie an einen offenen Array-Parameter ein Argument übergeben, das kein Array ist.

```
program Produce;
var
  P: ^Integer;
  I: Integer;
begin
  Writeln(P[I]);
end.
```

Wir versuchen, einen Index auf einen Zeiger auf integer anzuwenden – in C wäre dies zulässig, aber nicht in Delphi.

```
program Solve;
type
  TIntArray = array [0..MaxInt DIV sizeof(Integer)-1] of Integer;
var
  P: ^TIntArray;
  I: Integer;
begin
  Writeln(P^[I]);  (*Auch P[I] wäre korrekt*)
end.
```

In Delphi muss dem Compiler mitgeteilt werden, dass P auf ein Array von ganzen Zahlen zeigen soll.

4.1.3.6.323 Ausdruckstyp muss BOOLEAN sein (E2012)

Diese Fehlermeldung wird ausgegeben, wenn ein Ausdruck als Bedingung benutzt wird und daher vom Typ Boolean sein muss. Dies trifft für den Steuerausdruck der Anweisungen if, while und repeat zu sowie für den Ausdruck, mit dem ein bedingter Programmstopp gesteuert wird.

```
program Produce;
var
  P: Pointer;
begin
  if P then
    Writeln('P <> nil');
end.
```

Hier hat ein C++ Programmierer einfach eine Zeigervariable als Bedingung einer if-Anweisung benutzt.

```
program Solve;
var
  P: Pointer;
begin
  if P <> nil then
    Writeln('P <> nil');
end.
```

In diesem Fall müssen Sie in Delphi expliziter vorgehen.

4.1.3.6.324 Klassentyp erwartet (E2021)

In bestimmten Situationen erwartet der Compiler einen Klassentyp:

- Als Vorfahr eines Klassentyps.
- In der on-Klausel einer try-except-Anweisung.
- Als erstes Argument einer raise-Anweisung.
- Als letzten Typ eines forward-deklarierten Klassentyps.

```
program Produce;
begin
raise 'Dies würde in C++ funktionieren, jedoch nicht in Delphi';
end.

program Solve;
uses SysUtils;
begin
raise Exception.Create('Das Problem kann aber umgangen werden');
end.
```

4.1.3.6.325 Diese Form des Methodenaufrufs ist nur für Klassenmethoden erlaubt (E2076)

Sie haben eine normale Methode nur durch Angabe des Klassentyps, nicht aber der Instanz aufgerufen.

Dies ist nur bei Klassenmethoden und Konstruktoren, nicht aber bei normalen Methoden und Destruktoren zulässig.

```
program Produce;

type
  TMyClass = class
  (*...*)
  end;
var
  MyClass : TMyClass;

begin
  MyClass := TMyClass.Create; (*OK, Konstruktor*)
  Writeln(TMyClass.ClassName); (*OK, Klassenmethode*)
  TMyClass.Destroy;           (*<-- Hier die Fehlermeldung*)
end.
```

In diesem Beispiel wird versucht, den Typ TMyClass freizugeben. Dies ist nicht sinnvoll und somit unzulässig.

```
program Solve;
type
  TMyClass = class
  (*...*)
  end;
var
  MyClass : TMyClass;

begin
  MyClass := TMyClass.Create; (*OK, Konstruktor*)
  Writeln(TMyClass.ClassName); (*OK, Klassenmethode*)
  MyClass.Destroy;           (*OK, für Instanz aufgerufen*)
end.
```

Wie Sie sehen, sollte eigentlich die Instanz und nicht der Typ selbst freigeben werden.

4.1.3.6.326 Klasse besitzt keine Standardeigenschaft (E2149)

Sie haben eine Klassen-Instanz-Variable in einem Array-Ausdruck benutzt, aber der Klassentyp hat keine Standard-Array-Eigenschaft deklariert.

```
program Produce;

type
  Base = class
  end;

var
  b : Base;

procedure P;
  var ch : Char;
begin
  ch := b[1];
end;

begin
end.
```

In diesem Beispiel wird eine Fehlermeldung ausgelöst, da Base keine Array-Eigenschaft deklariert und b selbst kein Array ist.

```
program Solve;

type
  Base = class
    function GetChar(i : Integer): Char;
    property data[i : Integer]: Char read GetChar; default;
  end;

var
  b : Base;

function Base.GetChar(i : Integer): Char;
begin GetChar := 'A';
end;

procedure P;
  var ch : Char;
begin
  ch := b[1];
  ch := b.data[1];
end;

begin
end.
```

Wenn Sie eine Standard-Eigenschaft für eine Klasse deklariert haben, können Sie die Klassen-Instanz-Variable in Array-Ausdrücken verwenden, als wäre die Klassen-Instanz-Variable selbst ein Array. Eine andere Möglichkeit liegt darin, die Bezeichnung der Eigenschaft für den eigentlichen Array-Zugriff zu benutzen.

Anmerkung: Wenn die Hinweisfunktion aktiviert ist, erhalten Sie zwei Warnmeldungen mit dem Inhalt, dass der an ch zugewiesene Wert niemals benutzt wird.

4.1.3.6.327 Feld- oder Methodenbezeichner erwartet (E2168)

Sie haben einen Bezeichner für die read- oder write-Klausel einer Eigenschaft angegeben, die kein Feld und keine Methode ist.

```
program Produce;

var
  r : string;

type
  Base = class
    t : string;
    property Title: string read Title write Title;
    property Caption : string read r write r;
  end;

begin
end.
```

Beide Eigenschaften in diesem Beispiel verursachen einen Fehler. Im ersten Fall liegt der Fehler darin, dass es nicht erlaubt ist, die Eigenschaft selbst als Lese- und Schreibmethode anzugeben. Im zweiten Fall ist r kein Element der Basisklasse.

```
program Solve;

type
  Base = class
    t : string;
    property Title: string read t write t;
  end;
```

```
begin
end.
```

Stellen Sie sicher, dass alle read- und write-Klauseln für Eigenschaften einen gültigen Feld- oder Methodenbezeichner angeben, der ein Element der Klasse ist, zu der die Eigenschaft gehört.

4.1.3.6.328 Unterstützender Klassentyp erforderlich (E2022)

Beim Deklarieren eines unterstützenden Klassentyps mit einer ancestor-Klausel muss der Vorfahrtyp eine unterstützende Klasse sein.

4.1.3.6.329 Statische Instanz- oder Klassenmethode erwartet (E2380)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.330 Ausdruckstyp muss BOOLEAN sein (E2012) (E2013)

Diese Fehlermeldung wird nur angezeigt, wenn der Konstantenausdruck, der die Anzahl von Zeichen in einem String-Typ angibt, keine ganze Zahl ist.

```
program Produce;
type
  color = (red,green,blue);
var
  S3 : string[Succ(High(color))];
begin
end.
```

Im diesem Beispiel wird versucht, die Anzahl von Elementen in einem String als abhängig vom größten Element vom Typ Farbe festzulegen – leider ist der Elementzähler unzulässigerweise vom Typ Farbe.

```
program Solve;
type
  color = (red,green,blue);
var
  S3 : string[ord(High(color))+1];
begin
end.
```

4.1.3.6.331 Interface-Typ benötigt (E2205)

Es wurde ein Schnittstellentyp erwartet, aber nicht gefunden. Häufig beruht dieser Fehler auf der Angabe eines benutzerdefinierten Typs, der nicht als Schnittstellentyp deklariert wurde.

```
program Produce;
type
  Name = string;

  MyObject = class
  end;

  MyInterface = interface(MyObject)
  end;

  Base = class(TObject, Name)
  end;

begin
end.
```

In diesem Beispiel ist die Deklaration des Typs Base falsch, weil Name nicht als Schnittstellentyp deklariert ist. Dasselbe gilt für MyInterface, weil dessen Vorfahrschnittstelle nicht als solche deklariert wurde.

```
program Solve;
type
  BaseInterface = interface
  end;

  MyInterface = interface(BaseInterface)
  end;

  Base = class(TObject, MyInterface)
  end;

begin
end.
```

Bei Auftreten dieses Fehlers überprüfen Sie am besten noch einmal den Quelltext, um festzustellen, was wirklich beabsichtigt war. Soll eine Klasse eine Schnittstelle implementieren, muss sie zuerst explizit von einer Basisklasse wie TObject abgeleitet werden. Bei einer Erweiterung können Schnittstellen nur eine einzige Schnittstelle als Vorfahr haben.

In diesem Beispiel wird die Schnittstelle korrekt von einer anderen Schnittstelle abgeleitet und die Objektdefinition gibt richtigerweise eine Basis an, so dass Schnittstellen festgelegt werden können.

4.1.3.6.332 Label erwartet (E2031)

Diese Fehlermeldung tritt auf, wenn der Bezeichner, der in einer goto-Anweisung angegeben oder als Label bei der Inline-Assemblierung verwendet wird, nicht als Label deklariert wurde.

```
program Produce;

begin
  if 2*2 <> 4 then
    goto Exit; (*--- Hier die Fehlermeldung Exit ist ebenfalls eine Standard-Prozedur*)
  (*...*)
Exit:           (*Hier weitere Fehlermeldungen*)
end.

  program Solve;
label
  Exit;           (*Labels müssen in Delphi deklariert werden*)
begin
  if 2*2 <> 4 then
    goto Exit;
  (*...*)
Exit:
end.
```

4.1.3.6.333 Diese Form des Methodenaufrufs ist nur in Methoden von abgeleiteten Typen erlaubt (E2075)

Diese Fehlermeldung tritt auf, wenn in einer Routine, die keine Methode ist, eine Methode eines abgeleiteten Typs aufgerufen wird.

```
program Produce;

type
  TMyClass = class
    constructor Create;
  end;

  procedure Create;
begin
```

```

    inherited Create;      (*<-- Hier die Fehlermeldung*)
end;

begin
end.

In diesem Beispiel wird der geerbte Konstruktor in der Prozedur Create aufgerufen, die aber keine Methode ist.

program Solve;

type
  TMyClass = class
    constructor Create;
  end;

constructor TMyClass.Create;
begin
  inherited Create;
end;

begin
end.

```

Vergewissern Sie sich, dass Sie diese Art von Aufruf nur in einer Methode durchführen.

4.1.3.6.334 Objekttyp erforderlich (E2019)

Diese Fehlermeldung wird angezeigt, wenn der Compiler einen Objekttyp erwartet. Beispielsweise muss der ursprüngliche Typ eines Objekts ebenfalls ein Objekttyp sein.

```

type
  MyObject = object(TObject)
  end;
begin
end.

```

In der Unit System ist TObject ein Klassentyp; die Ableitung eines Objekttyps ist daher nicht möglich.

```

program Solve;
type
  MyObject = class (*Tatsächlich bedeutet dies: class(TObject)*)
  end;
begin
end.

```

Achten Sie darauf, dass der Typbezeichner wirklich für einen Objekttyp steht – möglicherweise ist er falsch geschrieben oder wird von einem Bezeichner einer anderen Unit verborgen.

4.1.3.6.335 Objekt oder Klassentyp erforderlich (E2020)

Diese Fehlermeldung wird angezeigt, wenn die Syntax Typename.Methodename verwendet wird, die Typbezeichnung sich aber weder auf ein Objekt noch auf einen Klassentyp bezieht.

```

program Produce;
type
  TInteger = class
    Value: Integer;
  end;
var
  V : TInteger;
begin
  V := Integer.Create;
end.

```

Der Typ Integer verfügt natürlich nicht über den Konstruktor Create, dafür aber TInteger.

```
program Solve;
type
  TInteger = class
    Value: Integer;
  end;
var
  V : TInteger;
begin
  V := TInteger.Create;
end.
```

Achten Sie darauf, dass der Bezeichner sich wirklich auf ein Objekt oder auf einen Klassentyp bezieht – möglicherweise ist er falsch geschrieben oder wird von einem Bezeichner einer anderen Unit verborgen.

4.1.3.6.336 Die überladene Prozedur '%s' muss mit der Direktive 'overload' gekennzeichnet sein (E2254)

Der Compiler hat eine Prozedur angetroffen, die nicht mit overload gekennzeichnet ist, aber den gleichen Namen wie eine bereits mit dieser Direktive gekennzeichnete Prozedur aufweist. Alle überladenen Prozeduren müssen entsprechend gekennzeichnet sein.

```
program Produce;

procedure f0(a : integer); overload;
begin
end;

procedure f0(a : integer; ch : char);
begin
end;

begin
end.
```

The procedure f0(a : integer; ch : char) causes the error since it is not marked with the overload keyword.

```
program solve;

procedure f0(a : integer); overload;
begin
end;

procedure f0(a : integer; ch : char); overload;
begin
end;

begin
end.
```

Wenn es sich bei der Prozedur um eine überladene Prozedur handeln soll, müssen Sie sie als überladen ausweisen. Andernfalls ändern Sie den Namen.

4.1.3.6.337 Zeigertyp erforderlich (E2017)

Diese Fehlermeldung wird angezeigt, wenn Sie den dereferenzierenden Operator "^" auf einen Operanden anwenden, der kein Zeiger ist, und (ein Sonderfall) wenn der zweite Operand in einer Anweisung "Raise <exception> at <address>" kein Zeiger ist.

```
program Produce;
var
  C: TObject;
```

```
begin
  C^.Destroy;
end.
```

Obwohl Klassentypen intern als Zeiger auf die eigentlichen Informationen implementiert werden, ist es nicht zulässig, den dereferenzierenden Operator auf Klassentypen auf Quellebene anzuwenden. Dies ist außerdem nicht notwendig – der Compiler wird automatisch eine Dereferenzierung durchführen, wenn diese erforderlich ist.

```
program Solve;
var
  C: TObject;
begin
  C.Destroy;
end.
```

Geben Sie einfach den dereferenzierenden Operator nicht an. Der Compiler führt dann automatisch die richtige Operation durch.

4.1.3.6.338 Bei der vorherigen Deklaration von '%s' wurde die Direktive 'overload' nicht angegeben (E2267)

Es gibt zwei Lösungen für dieses Problem. Sie können entweder den Überladungsversuch entfernen oder die Anweisung overload in die Originaldeklaration aufnehmen. Dieses Beispiel zeigt die zweite Möglichkeit.

```
program Produce;
type
  Base = class
    procedure func(a : integer);
    procedure func(a : char); overload;
  end;

  procedure Base.func(a : integer);
begin
end;

  procedure Base.func(a : char);
begin
end;

end.
```

In diesem Beispiel wird versucht, die char-Version der Funktion func zu überladen, ohne die erste Version von func mit overload als überladbar zu markieren.

Sie müssen alle Funktionen, die überladen werden sollen, mit der overload-Direktive kennzeichnen. Wenn overload nicht für alle Versionen erforderlich wäre, wäre es möglich, einen neuen Methode einzuführen, die eine vorhandene Methode überlädt. Durch eine Neucompilierung würde dann eine anderes Verhalten erreicht.

```
program Solve;
type
  Base = class
    procedure func(a : integer); overload;
    procedure func(a : char); overload;
  end;

  procedure Base.func(a : integer);
begin
end;

  procedure Base.func(a : char);
begin
end;

end.
```

4.1.3.6.339 Prozedur- oder Funktionsname erforderlich (E2121)

Sie haben in einer exports-Klausel einen Bezeichner angegeben, der keine Prozedur oder Funktion darstellt.

```
library Produce;

var
  y: procedure;

exports y;
begin
end.
```

Aus einer integrierten Bibliothek können keine Variablen exportiert werden. Dies gilt auch dann, wenn sie vom Typ procedure sind.

```
program Solve;

procedure ExportMe;
begin
end;

exports ExportMe;
begin
end.
```

Stellen Sie sicher, dass alle Bezeichner in einer exports-Klausel wirklich Prozeduren darstellen.

4.1.3.6.340 Eigenschaft erforderlich (E2299)

Sie müssen dem Programm eine Eigenschaft hinzufügen.

Die Deklaration einer Eigenschaft muss einen Namen, einen Typ und mindestens eine Zugriffsangabe enthalten. Die Syntax lautet folgendermaßen:

```
property Eigenschaftsname[ Indizes]: typeindex Integer-Konstante Bezeichner;
```

Die Variablen haben folgende Bedeutung:

- Eigenschaftsname ist ein beliebiger gültiger Bezeichner
- [Indizes] ist optional und besteht aus einer Folge von durch Strichpunkte getrennten Parameterdeklarationen
- Jede Deklaration hat die Form Bezeichner1, ..., Bezeichner: type
- Typ muss ein vordefinierter oder vorher deklarierter Typbezeichner sein. Eigenschaftsdeklarationen der Form property Num: 0..9 ... sind somit unzulässig.
- Die Indexklausel Integer-Konstante ist optional.
- Bezeichner ist eine Folge von read, write, stored, default (oder nodefault) und implements

Jede Eigenschaftsdeklaration muss zumindest einen read- oder write-Bezeichner enthalten.

Weitere Informationen finden Sie im Hilfethema 'Eigenschaften'.

4.1.3.6.341 Record, Objekt oder Klassentyp erforderlich (E2018)

Der Compiler erwartete, die Typbezeichnung zu finden, mit dem ein Datensatz, ein Objekt oder eine Klasse festgelegt wurde, konnte jedoch keine Typbezeichnung finden.

```
program Produce;
```

```

type
  RecordDesc = class
    ch : Char;
  end;

var
  pCh : PChar;
  r : RecordDesc;

procedure A;
begin
  pCh.ch := 'A';      (* Fall 1 *)

  with pCh do begin (* Fall 2 *)
  end;
end;
end.

```

Für diesen Fehler gibt es zwei mögliche Ursachen: Im ersten Fall besteht der Fehler in der Anwendung von '!' auf ein Objekt, das kein Datensatz ist. Im zweiten Fall resultiert der Fehler aus der Verwendung einer Variablen, die nicht den richtigen Typ für eine WITH-Anweisung hat.

```

program Solve;

type
  RecordDesc = class
    ch : Char;
  end;

var
  r : RecordDesc;

procedure A;
begin
  r.ch := 'A';      (* Fall 1 *)

  with r do begin (* Fall 2 *)
  end;
end;
end.

```

Die einfachste Möglichkeit zur Behebung dieses Fehlers besteht darin, das Symbol "." und WITH nur für Datensatz-, Objekt- und Klassenvariablen zu verwenden.

4.1.3.6.342 Funktion benötigt Ergebnistyp (E2023)

Sie haben eine Funktion deklariert, aber keinen Rückgabetyp festgelegt.

```

program Produce;

function Sum(A: array of Integer);
var I: Integer;
begin
  Result := 0;
  for I := 0 to High(A) do
    Result := Result + A[I];
end;

begin
end.

```

Hier soll Sum eine Funktion sein, allerdings haben wir den Compiler darüber nicht unterrichtet.

```
program Solve;
```

```

function Sum(A: array of Integer): Integer;
var I: Integer;
begin
  Result := 0;
  for I := 0 to High(A) do
    Result := Result + A[I];
end;

begin
end.

```

Achten Sie einfach darauf, dass Sie einen Ergebnistyp festlegen.

4.1.3.6.343 Globale Prozedur oder statische Klassenmethode erwartet (E2366)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.344 Variable erforderlich (E2036)

Diese Fehlermeldung tritt auf, wenn Sie versuchen, die Adresse eines Ausdrucks oder einer Konstanten zu benutzen.

```

program Produce;
var
  I: Integer;
  PI: ^Integer;
begin
  PI := Addr(1);
end.

```

Eine Konstante wie 1 hat keine Speicheradresse; Sie können also weder den Operator noch die Standard-Funktion Addr auf diese Konstante anwenden.

```

program Solve;
var
  I: Integer;
  PI: ^Integer;
begin
  PI := Addr(I);
end.

```

Sie müssen darauf achten, dass Sie die Adresse der Variablen benutzen.

4.1.3.6.345 **TYPEOF kann nur auf Objekttypen mit VMT angewendet werden (E2082)**

Diese Fehlermeldung wird angezeigt, wenn Sie die Standardfunktion TypeOf auf einen Objekttyp anwenden, der über keine virtuelle Methodentabelle verfügt.

Eine einfache Abhilfe besteht darin, eine virtuelle Dummy-Prozedur zu deklarieren, damit der Compiler eine VMT erzeugt.

```

program Produce;

type
  TMyObject = object
    procedure MyProc;
  end;

procedure TMyObject.MyProc;
begin
  (*...*)

```

```

end;

var
  P: Pointer;
begin
  P := TypeOf(TMyObject);      (*<-- Hier die Fehlermeldung*)
end.

```

In diesem Beispiel wird die Standardfunktion `TypeOf` auf den Typ `TMyObject` angewendet, der keine virtuellen Funktionen und somit keine VMT hat.

```

program Solve;

type
  TMyObject = object
    procedure MyProc;
    procedure Dummy; virtual;
  end;

procedure TMyObject.MyProc;
begin
  (*...*)
end;

procedure TMyObject.Dummy;
begin
end;

var
  P: Pointer;
begin
  P := TypeOf(TMyObject);
end.

```

Deklarieren Sie eine virtuelle `Dummy`-Funktion, oder entfernen Sie den Aufruf von `TypeOf`.

4.1.3.6.346 Anweisung erforderlich, aber Ausdruck vom Typ '%s' gefunden (E2014)

Der Compiler erwartete, eine Anweisung zu finden, stieß jedoch auf einen Ausdruck des angegebenen Typs.

```

program Produce;
  var
    a: Integer;
begin
  (3 + 4);
end.

```

In diesem Beispiel erwartet der Compiler, eine Anweisung zu finden, beispielsweise `if`, `while` oder `repeat`, stößt jedoch auf den Ausdruck `(3+4)`.

```

program Produce;
  var
    a: Integer;
begin
  a := (3 + 4);
end.

```

Die Lösung hier ist die Zuweisung des Ergebnisses des Ausdrucks `(3+4)` zur Variable `a`. Eine andere Lösung wäre, den betreffenden Ausdruck aus dem Quelltext zu entfernen. Die Entscheidung hängt jeweils von der Situation ab.

4.1.3.6.347 Zu viele verschachtelte bedingte Symbole (E2279)

Direktiven zur bedingten Compilierung können bis zu 32 Ebenen tief verschachtelt werden.

4.1.3.6.348 Der voll qualifizierte, verschachtelte Typname %s übersteigt die 1024 Byte-Grenze (E2409)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.349 Prozedur NEW benötigt einen Konstruktor (E2079)

Diese Fehlermeldung wird angezeigt, wenn Sie in der Parameterliste einen Bezeichner an New übergeben, der kein Konstruktor ist.

```
program Produce;

type
  PMyObject = ^TMyObject;
  TMyObject = object
    F: Integer;
  constructor Init;
  destructor Done;
  end;

constructor TMyObject.Init;
begin
  F := 42;
end;

destructor TMyObject.Done;
begin
end;

var
  P: PMyObject;

begin
  New(P, Done);           (*<-- Hier die Fehlermeldung*)
end.
```

New wurde versehentlich mit dem Destruktor anstelle des Konstruktors aufgerufen.

```
program Solve;

type
  PMyObject = ^TMyObject;
  TMyObject = object
    F: Integer;
  constructor Init;
  destructor Done;
  end;

constructor TMyObject.Init;
begin
  F := 42;
end;

destructor TMyObject.Done;
begin
end;
```

```

var
  P: PMyObject;

begin
  New(P, Init);
end.

```

Vergewissern Sie sich, dass Sie der Standardfunktion New nur den Konstruktor oder kein zweites Argument übergeben.

4.1.3.6.350 Ein NeverBuild-Package '%s' benötigt ein AlwaysBuild-Package '%s' (E2220)

4

Sie versuchen ein NoBuild-Package zu erstellen, das ein AlwaysBuild-Package anfordert. Da die Schnittstelle eines AlwaysBuild-Package sich jederzeit ändern kann, und der Compiler bei Angabe des Flags NoBuild davon ausgeht, dass das Package auf dem neuesten Stand ist, kann ein NoBuild-Package nur solche Packages anfordern, die auch mit NoBuild markiert sind.

```

package Base;
end.

(*$IMPLICITBUILD OFF*)
package NoBuild;
  requires Base;
end.

```

In diesem Beispiel fordert das NoBuild-Package ein Package an, das im AlwaysBuild-Status compiliert wurde.

```

(*$IMPLICITBUILD OFF*)
package Base;
end.

(*$IMPLICITBUILD OFF*)
package NoBuild;
  requires Base;
end.

```

Zur Lösung des Problems wird Base in ein NeverBuild-Package umgewandelt. Alternativ könnten Sie auch {\$IMPLICITBUILD OFF} aus dem NoBuild-Package löschen, und das Package in ein AlwaysBuild-Package umwandeln.

4.1.3.6.351 Durch das Überschreiben erhält die virtuelle Methode '%s.%s' eine geringere Sichtbarkeit (%s) als die Basisklasse '%s' (%s) (x2269)

Der in der Fehlermeldung genannte Methode wurde als Überschreibung einer virtuellen Methode in einer Basisklasse deklariert, aber die Sichtbarkeit in der aktuellen Klasse ist geringer als diejenige, die für die Basisklasse der gleichen Methode festgelegt wurde.

Obwohl die Sichtbarkeitsregeln von Delphi anscheinend angeben, dass die Funktion niemals sichtbar ist, bewirken die Regeln zum Aufrufen von virtuellen Funktionen, dass die Funktion durch einen virtuellen Aufruf korrekt aufgerufen wird.

In der Regel tritt dieser Fehler auf, wenn eine Methode in einem als private oder protected deklarierten Abschnitt der abgeleiteten Klasse deklariert wird, aber in einem als protected oder public/published deklarierten Abschnitt in der Basisklasse auftaucht.

```

unit Produce;
interface

type
  Base = class(TObject)
  public

```

```

procedure VirtualProcedure(X: Integer); virtual;
end;

Extended = class (Base)
protected
  procedure VirtualProcedure(X: Integer); override;
end;

implementation

procedure Base.VirtualProcedure(X: Integer);
begin
end;

procedure Extended.VirtualProcedure(X: Integer);
begin
end;
end.

```

Das Beispiel zeigt, wie dieses Problem durch das Einfügen von Extended.VirtualProcedure in den als protected deklarierten Abschnitt entsteht.

In der Praxis ist dies zwar nie schädlich, aber es kann jemanden verwirren, der die Dokumentation liest die Sichtbarkeitsattribute betrachtet.

Dieser Hinweis wird nur für Klassen produziert, die in dem interface-Abschnitt einer Unit enthalten sind.

```

unit Solve;
interface

type
  Base = class(TObject)
  public
    procedure VirtualProcedure(X: Integer); virtual;
  end;

  Extended = class (Base)
  public
    procedure VirtualProcedure(X: Integer); override;
  end;

implementation

procedure Base.VirtualProcedure(X: Integer);
begin
end;

procedure Extended.VirtualProcedure(X: Integer);
begin
end;
end.

```

Es gibt für dieses Problem drei Lösungsmöglichkeiten:

1. Den Hinweis ignorieren.
2. Die Sichtbarkeit in der abgeleiteten Klasse so ändern, dass sie der Sichtbarkeit in der Basisklasse entspricht.
3. Die Klassendefinition in den Implementierungsabschnitt verschieben.

Im Beispiel wurde die Sichtbarkeit der Methode an die Sichtbarkeit in der Basisklasse angepasst.

4.1.3.6.352 Unit %s in Package %s verweist auf Unit %s, die in keinem Package vorhanden ist. Package-Unit dürfen nur auf Package-Units verweisen (E2411)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.353 Konstruktoren und Destruktoren müssen die %s-Aufrufkonvention haben (E2236)

Es wurde versucht, die voreingestellte Aufrufkonvention eines Konstruktors oder Destruktors zu ändern.

```
program Produce;

type
  TBase = class
    constructor Create; pascal;
  end;

  constructor TBase.Create;
begin
end;

begin
end.
  program Solve;

type
  TBase = class
    constructor Create;
  end;

  constructor TBase.Create;
begin
end;

begin
end.
```

Dieses Problem lässt sich nur (wie im Beispiel oben) durch das Entfernen der fehlerhaften Aufrufkonvention aus der Konstrukt- bzw. Destruktordefinition lösen.

4.1.3.6.354 Im OLE-Automatisierungsbereich sind nur register-Aufrufkonventionen zulässig (E2179)

Sie haben eine unzulässige Aufrufkonvention für eine Methode festgelegt, die im Abschnitt automated einer Klassendeklaration erscheint.

```
program Produce;

type
  Base = class
    automated
      procedure Method; cdecl;
    end;

    procedure Base.Method; cdecl;
begin
end;
```

begin
end.

Die Sprachenspezifikation lässt im OLE-Automatisierungsabschnitt keine Aufrufkonventionen außer register zu. Im obigen Quelltext lautet die betreffende Anweisung 'cdecl'.

```
program Solve;

type
  Base = class
  automated
    procedure Method; register;
    procedure Method2;
  end;

  procedure Base.Method; register;
begin
end;

  procedure Base.Method2;
begin
end;

begin
end.
```

Für diesen Fehler gibt es drei Lösungsmöglichkeiten. Erstens: Es werden kein Aufrufkonventionen für Methoden festgelegt, die in einem Automatisierungsabschnitt deklariert sind. Zweitens: Es wird nur die register-Aufrufkonvention festgelegt. Drittens: Die betreffende Deklaration wird aus dem Automatisierungsabschnitt entfernt.

4.1.3.6.355 Publizierte Methoden zum Setzen und Lesen von Eigenschaften müssen die %s-Aufrufkonvention besitzen (E2270)

Eine Eigenschaft, die in einem als published deklarierten Abschnitt enthalten ist, besitzt eine Zugriffsmethode (getter oder setter), die keine korrekte Aufrufkonvention besitzt.

```
unit Produce;
interface
  type
    Base = class
    public
      function getter : Integer; cdecl;
    published
      property Value : Integer read getter;
    end;

  implementation
  function Base.getter : Integer;
  begin getter := 0;
  end;

end.
```

Im Beispiel wurde die Getter-Methode Getter der Eigenschaft Value mit der cdecl-Aufrufkonvention deklariert. Da Value als published deklariert ist, erzeugt dieses Beispiel einen Fehler.

```
unit Solve;
interface
  type
    Base = class
    public
      function getter : Integer;
    published
```

```

  property Value : Integer read getter;
end;

implementation
function Base.getter : Integer;
begin getter := 0;
end;

end.

```

Die Lösung besteht darin, für den Getter die korrekte Aufrufkonvention zu deklarieren (dies ist der Standard). In diesem Beispiel wurde keine Aufrufkonvention festgelegt.

4.1.3.6.356 Potentiell polymorphe Konstruktoraufrufe müssen virtuell sein (E2391)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.357 Label '%s' ist in der aktuellen Prozedur nicht deklariert (E2093)

Im Gegensatz zu Standard Pascal sind in Borland Delphi keine goto-Anweisungen erlaubt, die aus der Prozedur herausführen.

Solche Konstrukte werden gern zur Fehlerbehandlung eingesetzt. In Delphi steht für die Behandlung von Fehlern ein umfassenderer und strukturierter Mechanismus zur Verfügung, die so genannte Exception-Behandlung.

```

program Produce;

label 99;

procedure MyProc;
begin
  (*Ein Fehler tritt auf...*)
  goto 99;
end;

begin
  MyProc;
  99:
  Writeln('Schwerwiegender Fehler');
end.

```

In diesem Beispiel wird versucht, die Verarbeitung durch eine aus der Prozedur MyProc herausführende goto-Anweisung zu unterbrechen.

```

program Solve;

uses SysUtils;

procedure MyProc;
begin
  (*Ein Fehler tritt auf...*)
  raise Exception.Create('Schwerwiegender Fehler');
end;

begin
  try
    MyProc;
  except
    on E: Exception do Writeln(E.Message);
  end;
end.

```

Hier wird der Programmablauf durch die Exception-Behandlung unterbrochen. Dabei kann zugleich eine Fehlermeldung übergeben werden. Eine weitere Möglichkeit besteht in der Verwendung der Standardprozeduren Halt oder RunError.

4.1.3.6.358 Der linken Seite kann nichts zugewiesen werden (E2064)

Dieser Fehler tritt auf, wenn ein Nur-Lesen-Objekt wie eine Konstante, ein Konstantenparameter oder der Rückgabewert einer Funktion modifiziert wird.

```
program Produce;

const
  c = 1;

procedure p(const s: string);
begin
  s := 'changed';           (*<-- Hier die Fehlermeldung*)
end;

function f: PChar;
begin
  f := 'Hello';            (*Das ist OK - nun wird der Rückgabewert gesetzt*)
end;

begin
  c := 2;                  (*<-- Hier die Fehlermeldung*)
  f := 'h';                (*<-- Hier die Fehlermeldung*)
end.
```

In diesem erfolgt eine Zuweisung an einen Konstantenparameter, eine Konstante und einen Rückgabewert einer Funktion. Keine dieser Operationen ist zulässig.

```
program Solve;

var
  c : Integer = 1;          (*Verwenden Sie eine initialisierte Variable*)

procedure p(const s: string);
begin
  s := 'changed';           (*Verwenden Sie einen var-Parameter*)
end;

function f: PChar;
begin
  f := 'Hello';             (*Das ist OK - nun wird der Rückgabewert gesetzt*)
end;

begin
  c := 2;
  f^ := 'h';                (*Das Programm wird zwar compiliert, stürzt aber zur Laufzeit ab*)
end.
```

Diese Probleme können auf zwei Arten gelöst werden. Ändern Sie entweder die Definition des Elements, das zugewiesen wird, damit die Operation zulässig wird, oder entfernen Sie die Zuweisung.

4.1.3.6.359 '%s' ist kein Name einer Unit (E2242)

Der Direktive \$NOINCLUDE muss ein bekannter Unit-Name zugewiesen werden.

4.1.3.6.360 for-in-Anweisung arbeitet nicht mit Kollektionstyp '%s' (E2430)

Eine for-in-Anweisung kann nur mit den folgenden Kollektionstypen zusammenarbeiten:

- Primitive Typen, die der Compiler erkennt, wie z.B. Arrays, Mengen oder Strings
 - Typen, die `IEnumerable` implementieren
 - Typen, die das `GetEnumerator`-Pattern implementieren (siehe Delphi-Sprachreferenz)
- Stellen Sie sicher, dass der angegebene Typ diesen Anforderungen entspricht.

Siehe auch

Deklarationen und Anweisungen (siehe Seite 862)

4.1.3.6.361 Es wurden keine Konfigurationsdateien gefunden (W1039)

Der Compiler konnte die Konfigurationsdateien nicht finden, auf die im Quelltext verwiesen wird.

4.1.3.6.362 Dispose wird für dynamische Arrays nicht unterstützt (da nicht erforderlich) (E2256)

Der Compiler hat festgestellt, dass die Standardprozedur `Dispose` für ein dynamisches Array verwendet wurde. Dynamische Arrays arbeiten mit einem Referenzzähler und werden automatisch freigegeben, wenn keine Referenzen mehr darauf vorhanden sind.

```
program Produce;
  var
    arr : array of integer;

begin
  SetLength(arr, 10);
  Dispose(arr);
end.
```

Die Verwendung von `Dispose` für das dynamische Array `Arr` verursacht den Fehler in diesem Beispiel.

```
program Produce;
  var
    arr : array of integer;

begin
  SetLength(arr, 10);
end.
```

Die einzige Lösung besteht darin, `Dispose` an dieser unzulässigen Stelle zu entfernen.

4.1.3.6.363 Es gibt keine überladene Version von '%s', die man mit diesen Argumenten aufrufen kann (E2250)

Es wurde versucht, eine überladene Funktion aufzurufen, die mit der aktuellen Gruppe überladener Prozeduren nicht zurückgeschrieben werden kann.

```
program Produce;

procedure f0(a : integer); overload;
begin
end;

procedure f0(a : char); overload;
begin
end;
```

```
begin
  f0(1, 2);
end.
```

Für die überladene Prozedur f0 gibt es zwei Versionen, eine übernimmt einen char-Parameter, die andere einen Integer-Parameter. Der Aufruf von f0 verwendet jedoch einen Gleitkommaparameter, der vom Compiler weder auf den Typ Char noch auf den Typ Integer zurückgeschrieben werden kann.

```
program Solve;

procedure f0(a : integer); overload;
begin
end;

procedure f0(a : char); overload;
begin
end;

begin
  f0(1);
end.
```

Für dieses Problem gibt es zwei Lösungen: Entweder geben Sie einen Parametertyp an, der in Übereinstimmung mit der überladenen Prozeduren zurückgeschrieben werden kann, oder Sie erzeugen eine neue Version der überladenen Prozedur mit einem Parameter des im Funktionsaufruf verwendeten Typs.

In diesem Beispiel wurde der Parametertyp so geändert, dass er mit einer der vorhandenen überladenen Versionen von f0 übereinstimmt.

4.1.3.6.364 Keine überladene Version der Array-Eigenschaft '%s' vorhanden, die mit diesen Argumenten verwendet werden kann (E2450)

Um diesen Fehler zu beheben, ändern Sie entweder die Argumente, so dass ihre Typen mit einer Version der Array-Eigenschaft übereinstimmen oder fügen Sie eine neue überladene Array-Eigenschaft mit Typen, die den Argumenten entsprechen, hinzu.

4.1.3.6.365 Es existiert keine überladene Version von '%s' mit dieser Parameterliste (E2273)

Es wurde versucht, eine überladene Prozedur aufzurufen. Es wurde aber keine geeignete Übereinstimmung gefunden.

```
program overload;
  procedure f(x : Char); overload;
  begin
  end;

  procedure f(x : Integer); overload;
  begin
  end;

begin
  f(1, 0);
end.
```

So wie in diesem Beispiel 'f' verwendet wird, kann der Compiler (mithilfe der Regeln für die Typkompatibilität und das Überladen) keine geeignete Übereinstimmung für den eigentlichen Parameter 1.0 finden.

```
program overload;
  procedure f(x : char); overload;
  begin
```

```

end;

procedure f(x : integer); overload;
begin
end;

begin
  f(1);
end.

```

Hier wurde der Aufruf von 'f' geändert und es wird ein Integer-Typ als eigentlicher Parameter übergeben. Dadurch kann der Compiler eine geeignete Übereinstimmung finden. Ein anderes Vorgehen zur Lösung dieses Problems wäre, einen neuen Prozedur anzulegen, die einen Gleitkommparameter übernimmt.

4.1.3.6.366 Prozedur kann keinen Ergebnistyp besitzen (E2025)

Sie haben eine Prozedur deklariert, ihr aber einen Ergebnistyp gegeben. Entweder wollten Sie eigentlich eine Funktion deklarieren, oder Sie sollten den Ergebnistyp löschen.

```

program Produce;

procedure DotProduct(const A,B: array of Double): Double;
var
  I: Integer;
begin
  Result := 0.0;
  for I := 0 to High(A) do
    Result := Result + A[I]*B[I];
end;

const
  C: array [1..3] of Double = (1,2,3);

begin
  Writeln( DotProduct(C,C) );
end.

```

Hier sollte DotProduct eigentlich eine Funktion sein. Es wurde nur aus Versehen das falsche Schlüsselwort angegeben.

```

program Solve;

function DotProduct(const A,B: array of Double): Double;
var
  I: Integer;
begin
  Result := 0.0;
  for I := 0 to High(A) do
    Result := Result + A[I]*B[I];
end;

const
  C: array [1..3] of Double = (1,2,3);

begin
  Writeln( DotProduct(C,C) );
end.

```

Achten Sie einfach darauf, einen Ergebnistyp festzulegen, wenn Sie eine Funktion deklarieren, und keinen Ergebnistyp, wenn Sie eine Prozedur deklarieren.

4.1.3.6.367 Rückgabewert der Funktion '%s' könnte undefiniert sein (W1035)

Diese Warnung wird angezeigt, wenn dem Rückgabewert einer Funktion nicht in jedem Codepfad ein Wert zugewiesen wurde.

Die Funktion könnte auch ausgeführt werden, ohne dass der Rückgabewert zugewiesen wird.

```
4
program Produce;
(*$WARNINGS ON*)
var
  B : Boolean;
  C: (Red,Green,Blue);

function Simple: Integer;
begin
end;                               (*<-- Hier die Warnmeldung*)

function IfStatement: Integer;
begin
  if B then
    Result := 42;
end;                               (*<-- Hier die Warnmeldung*)

function CaseStatement: Integer;
begin
  case C of
    Red..Blue: Result := 42;
  end;
end;                               (*<-- Hier die Warnmeldung*)

function TryStatement: Integer;
begin
  try
    Result := 42;
  except
    Writeln('Should not get here!');
  end;
end;                               (*<-- Hier die Warnmeldung*)

begin
  B := False;
end.
```

Das Problem bei IfStatement und CaseStatement besteht darin, dass der Rückgabewert (Result) nicht in jedem Quelltextpfad zugewiesen wird. Der Compiler vermutet, dass in TryStatement eine Exception vor der Zuweisung von Result auftreten kann.

```
program Solve;
(*$WARNINGS ON*)
var
  B : Boolean;
  C: (Red,Green,Blue);

function Simple: Integer;
begin
  Result := 42;
end;

function IfStatement: Integer;
begin
  if B then
    Result := 42
  else
    Result := 0;
end;

function CaseStatement: Integer;
begin
  case C of
    Red..Blue: Result := 42;
  else
    Result := 0;
  end;
end;
```

```

function TryStatement: Integer;
begin
  Result := 0;
  try
    Result := 42;
  except
    Writeln('Should not get here!');
  end;
end;

begin
  B := False;
end.

```

Die Lösung besteht darin, sicherzustellen, dass der Rückgabewert in jedem möglichen Quelltextpfad zugewiesen wird.

4.1.3.6.368 Typ '%s' besitzt keine Typinformation (E2134)

Sie wenden die Standardprozedur TypeInfo auf einen Typbezeichner an, dem keine Laufzeit-Typinformationen zugeordnet sind.

```

program Produce;

type
  Data = record
  end;

var
  v : Pointer;

begin
  v := TypeInfo(Data);
end.

```

Da Record-Typen keine Typinformationen generieren, ist diese Anweisung nicht erlaubt.

```

program Solve;

type
  Base = class
  end;

var
  v : Pointer;

begin
  v := TypeInfo(Base);
end.

```

Da Klassen RTTI generieren, ist dieser Aufruf von TypeInfo korrekt.

4.1.3.6.369 FOR oder WHILE Schleife wird nicht durchlaufen - gelöscht (H2135)

Der Compiler hat festgestellt, dass die angegebene Schleife nie durchlaufen wird. Daher wurde sie aus Optimierungsgründen entfernt. Beispiel:

```

program Produce;
(*$HINTS ON*)

var
  i : Integer;

begin
  i := 0;

```

```
WHILE FALSE AND (i < 100) DO
  INC(i);
end.
```

Der Compiler stellt fest, dass 'FALSE AND (i < 100)' immer False ergibt und die Schleife daher nie zur Ausführung kommt.

```
program Solve;
(*$HINTS ON*)
```

```
var
  i : Integer;

begin
  i := 0;
  WHILE i < 100 DO
    INC(i);
end.
```

Prüfen Sie, ob der Boolesche Ausdruck, der die Schleife steuert, nicht grundsätzlich False ergibt. Bei for-Schleifen sollten Sie darauf achten, dass Endwert minus Startwert immer größer oder gleich 1 ist.

Bei for-Schleifen wird diese Warnung angezeigt, wenn eine Longint-Schleifenvariable einen Wert annimmt, der außerhalb des Longint-Bereichs liegt. Beispiel:

```
var I: Cardinal;
begin
  For I := 0 to $FFFFFFFFF do
  ...
```

Sie können dieses Compiler-Problem beheben, indem Sie die for-Schleife durch eine while-Schleife ersetzen.

4.1.3.6.370 Lokale Klasse, Interface oder Objekttypen sind nicht erlaubt (E2059)

Klassen und Objekte können in einer Prozedur nicht lokal deklariert werden.

```
program Produce;

procedure MyProc;
type
  TMyClass = class
    Field: Integer;
  end;
begin
  (*...*)
end;
```

```
begin
end.
```

In MyProc wird eine Klasse lokal deklariert. Dies ist nicht zulässig.

```
program Solve;

type
  TMyClass = class
    Field: Integer;
  end;

procedure MyProc;
begin
  (*...*)
end;

begin
```

end.

Verlegen Sie einfach die Deklaration des Klassen- oder Objekttyps in den globalen Gültigkeitsbereich.

4.1.3.6.371 Bei der Compilierung zu Byte-Code können Objekttypen alten Stils nicht benutzt werden (E2248)

Objekttypen nach dem alten Stil sind bei der Compilierung zum Erzeugen von Byte-Code nicht zulässig.

4.1.3.6.372 Klasse, Interface und Objekttypen sind nur im Abschnitt type erlaubt (E2058)

Klassen- und Objekttypen müssen immer mit einer expliziten Typdeklaration in einem type-Abschnitt deklariert werden. Im Gegensatz dazu können Record-Typen anonym sein.

Der Hauptgrund besteht darin, dass Sie sonst nicht die Methoden dieses Typs deklarieren könnten.

```
program Produce;

var
  MyClass : class
    Field: Integer;
  end;

begin
end.
```

In diesem Beispiel wird ein Klassentyp in einer Variablendeklaration deklariert. Dies ist nicht zulässig.

```
program Solve;

type
  TMyClass = class
    Field: Integer;
  end;

var
  MyClass : TMyClass;

begin
end.
```

Die Lösung besteht darin, eine Typdeklaration für den Klassentyp einzufügen. Sie können aber auch den Klassen- in einen Record-Typ ändern.

4.1.3.6.373 Virtuelle Konstruktoren sind nicht erlaubt (E2062)

Objekttypen können im Gegensatz zu Klassentypen nur statische Konstruktoren haben.

```
program Produce;

type
  TMyObject = object
    constructor Init; virtual;
  end;

constructor TMyObject.Init;
begin
end;
```

```
begin
end.
```

In diesem Beispiel wird ein virtueller Konstruktor deklariert. Das ist aber bei Objekttypen nicht sinnvoll und daher unzulässig.

```
program Solve;

type
  TMyObject = object
    constructor Init;
  end;

constructor TMyObject.Init;
begin
end;

begin
end.
```

Definieren Sie entweder einen statischen Konstruktor, oder verwenden Sie einen Klassentyp im neuen Stil, der einen virtuellen Konstruktor haben kann.

4.1.3.6.374 Inline-Funktion darf kein Argument für ein Open Array haben (E2439)

Um diesen Fehler zu beheben, entfernen Sie die Direktive **inline** oder verwenden Sie einen explizit deklarierten, dynamischen Array-Typ anstelle des Open-Array-Arguments.

Siehe auch

Parameter (siehe Seite 938)

Strukturierte Typen (siehe Seite 902)

4.1.3.6.375 Wert '%s' für Option %s wurde abgeschnitten (W1049)

String-basierte Compiler-Optionen, wie z.B. Unit-Suchpfade, haben eine begrenzte Puffergröße.

Diese Meldung gibt an, dass die Puffergröße überschritten wurde.

4.1.3.6.376 Ordinaltyp erforderlich (E2001)

Der Compiler benötigt an dieser Stelle einen Ordinaltyp. Ordinaltypen sind die vordefinierten Integer-, Char-, WideChar- und Boolesche Typen, sowie deklarierte Aufzählungstypen.

Ordinaltypen werden in verschiedenen Situationen erwartet:

- Der Indextyp eines Arrays muss ein Ordinaltyp sein.
- Die Ober- und Untergrenze eines Untermengentyps müssen konstante Ausdrücke eines ordinalen Typs sein.
- Der Elementtyp einer Menge muss ein Ordinaltyp sein.
- Der Auswahl-Ausdruck einer case-Anweisung muss ein Ordinaltyp sein.
- Das erste Argument der Standard-Prozeduren Inc und Dec muss eine Variable vom ein Ordinaltyp oder ein Zeiger sein.

```
program Produce;
type
  TByteSet = set of 0..7;
var
  BitCount: array [TByteSet] of Integer;
```

```
begin
end.
```

Der Indextyp eines Arrays muss ein Ordinaltyp sein – der Typ TByteSet ist eine Menge, kein Ordinaltyp.

```
program Solve;
type
  TByteSet = set of 0..7;
var
  BitCount: array [Byte] of Integer;
begin
end.
```

Geben Sie einen Ordinaltyp als Indextyp des Arrays an.

4.1.3.6.377 Methoden zum Setzen und Lesen von Eigenschaften dürfen nicht überladen werden (E2271)

Für eine Eigenschaft wurde eine überladene Prozedur entweder als ihre read- oder write-Methode angegeben.

```
unit Produce;
interface
  type
    Base = class
      public
        function getter : Integer; overload;
        function getter(a : char) : Integer; overload;
        property Value : Integer read getter;
    end;

  implementation
  function Base.getter : Integer;
  begin getter := 0;
  end;

  function Base.getter(a : char) : Integer;
  begin
  end;
end.
```

Im Beispiel wird ein Fehler generiert, weil die Methode getter überladen ist.

```
unit Solve;
interface
  type
    Base = class
      public
        function getter : Integer;
        property Value : Integer read getter;
    end;

  implementation
  function Base.getter : Integer;
  begin getter := 0;
  end;

end.
```

Die Lösung besteht darin, die unzulässige Überladung zu entfernen (wie im Beispiel oben).

4.1.3.6.378 Die Schreibweise der überschriebenen Methode %s.%s sollte mit der des Vorfahren %s.%s übereinstimmen (H2365)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.379 Methode '%s' nicht in Basisklasse gefunden (E2137)

Sie haben für eine Methode die Direktive override angegeben. Der Compiler kann jedoch in der Basisklasse keine Prozedur mit diesem Namen finden.

```
program Produce;

type
  Base = class
    procedure Title; virtual;
  end;

  Derived = class (Base)
    procedure Titl; override;
  end;

  procedure Base.Title;
begin
end;

procedure Derived.Titl;
begin
end;

begin
end.
```

Ein üblicher Grund für diesen Fehler ist ein Tippfehler im Quelltext. Stellen Sie sicher, dass der Name der override-Prozedur so geschrieben ist wie in der Basisklasse. Eine weitere Möglichkeit besteht darin, dass die Basisklasse die betreffende Prozedur nicht zur Verfügung stellt. In diesem Fall ist eine nähere Prüfung des Problems erforderlich.

```
program Solve;

type
  Base = class
    procedure Title; virtual;
  end;

  Derived = class (Base)
    procedure Title; override;
  end;

  procedure Base.Title;
begin
end;

procedure Derived.Title;
begin
end;

begin
end.
```

Die Lösung besteht in diesem Beispiel in der richtigen Schreibweise des Prozedurnamens in Derived.

4.1.3.6.380 Eine final-Methode kann nicht überschrieben werden (E2352)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.381 Eine nicht-virtuelle Methode kann nicht überschrieben werden (E2170)

Sie haben in einer abgeleiteten Klasse versucht, eine Basismethode zu überschreiben, die nicht als einer der virtuellen Typen deklariert war.

```
program Produce;

type
  Base = class
    procedure StaticMethod;
  end;

  Derived = class (Base)
    procedure StaticMethod; override;
  end;

  procedure Base.StaticMethod;
begin
end;

procedure Derived.StaticMethod;
begin
end;

begin
end.
```

In diesem Beispiel wird ein Fehler ausgelöst, weil Base.StaticMethod nicht als virtuelle Methode deklariert wurde und daher eine Überschreibung ihrer Deklaration nicht möglich ist.

```
program Solve;

type
  Base = class
    procedure StaticMethod;
  end;

  Derived = class (Base)
    procedure StaticMethod;
  end;

  procedure Base.StaticMethod;
begin
end;

procedure Derived.StaticMethod;
begin
end;

begin
end.
```

Die einzige Möglichkeit, diesen Fehler aus Ihrem Programm zu entfernen, wenn Sie nicht über den Quelltext für die Basisklassen verfügen, ist die Entfernung der override-Festlegung aus der Deklaration der abgeleiteten Methode. Wenn Sie über den Quelltext der Basisklasse verfügen, könnten Sie - nach sorgfältiger Überlegung - die Basismethode ändern und als virtuellen Typ deklarieren. Beachten Sie aber, dass diese Änderung eine drastische Auswirkung auf Ihre Programme haben kann.

4.1.3.6.382 Die exportierte Package-Thread-Variable '%s.%s' darf nicht außerhalb dieses Package verwendet werden (W1032)

Windows unterstützt das Exportieren von threadvar-Variablen aus einer DLL zwar nicht, aber da die Verwendung von Packages als semantisch gleichwertig mit der Compilierung eines Projekts ohne diese Packages definiert ist, muss der Delphi-Compiler dieses Konstrukt unterstützen.

Mit dieser Warnung werden Sie darüber informiert, dass Sie eine Unit mit einbezogen haben, die eine lokal gültige Schnittstellenvariable für einen Thread in einem Package verwendet. Da dies unzulässig ist, haben Sie aus einer Unit außerhalb des Package keinen Zugriff auf die Variable.

Ein entsprechender Zugriffsversuch ist nur scheinbar erfolgreich.

Eine Lösung besteht darin, die threadvar-Variable in den Implementierungsabschnitt zu verschieben und eine Funktion zur Verfügung zu stellen, die den Wert der Variablen abruft.

4.1.3.6.383 Falsches Package-Unit-Format: %s.%s (E2213)

Bei dem Versuch, die angegebene Unit aus dem Package zu laden, hat der Compiler festgestellt, dass die Unit beschädigt ist. Der Grund dafür ist möglicherweise ein unerwarteter Abbruch des Compilers beim Schreiben der Package-Datei (z. B. durch Stromausfall). In dieser Situation sollten Sie als Erstes die betreffende DCP-Datei löschen und das Package erneut compilieren.

4.1.3.6.384 Package '%s' kann nicht compiliert werden (F2220)

Bei der Compilierung des genannten Package ist ein Fehler aufgetreten. Hier gibt es nur eine Lösung: Korrigieren Sie den Fehler und compilieren Sie das Package erneut.

4.1.3.6.385 Die Packages '%s' und '%s' enthalten beide Unit '%s' (E2199)

Das Projekt, das Sie compilieren wollen, verwendet zwei Packages, die beide dieselbe Unit enthalten. Dies ist aber innerhalb eines einzelnen Projekts nicht zulässig, da der Compiler die daraus resultierende Doppeldeutigkeit nicht auflösen kann.

Meist liegt die Ursache für dieses Problem in einem unzulänglich definierten Package-Satz.

Die einzige Möglichkeit zur Lösung dieses Problems besteht im Neuentwurf der Package-Hierarchie, um die Doppeldeutigkeit zu beheben.

4.1.3.6.386 Package '%s' enthält bereits die Unit '%s' (E2200)

Das compilierte Package fordert (entweder über die requires-Klausel oder die Package-Liste) ein anderes Package an, das bereits die in der Meldung angegebene Unit enthält.

Es ist nicht zulässig, dass zwei miteinander verknüpfte Packages dieselbe Unit enthalten. Zur Lösung dieses Problem löschen Sie entweder die Unit aus einem der Packages oder heben die Verknüpfung zwischen den beiden Packages auf.

4.1.3.6.387 Package '%s' wird nicht auf Platte geschrieben, da Option -J aktiviert ist (W1031)

Der Compiler kann das Package nicht auf die Festplatte schreiben, da die Option -J versucht, eine Objektdatei zu erstellen.

4.1.3.6.388 'Never-build'-Package '%s' muss neu compiliert werden (E2225)

Das in der Meldung angegebene Package wurde als NeverBuild-Package compiliert. Dieses Package fordert aber ein weiteres Package an, dessen Schnittstelle geändert wurde. Das genannte Package kann ohne erneute Compilierung nicht verwendet werden, weil es mit einer anderen Schnittstelle des erforderlichen Package verbunden wurde.

Dieses Problem kann nur auf eine Art gelöst werden: Sie müssen das Fehler verursachende Package erneut manuell compilieren. Vergessen Sie nicht, die Option NeverBuild anzugeben (wenn dies noch gewünscht wird).

4.1.3.6.389 Package '%s' verwendet oder exportiert nicht '%s.%s' (H2235)

Eine Unit wurde in ein Package compiliert, in dem ein Symbol enthalten ist, das weder im interface-Abschnitt der Unit erscheint noch von einer anderen Anweisung innerhalb der Unit referenziert wird. Es handelt sich hier um "toten" Quelltext, der ohne Auswirkungen auf das Programm entfernt werden kann.

4.1.3.6.390 Für den Zugriff auf '%s' von Unit '%s' wird die Referenz auf importierte Daten (\$G) benötigt (E2201)

Bei der Compilierung der in der Meldung angegebenen Unit war der Schalter \$G nicht aktiviert.

```
(*$IMPORTEDDATA OFF*)
unit u0;
interface
implementation
begin
  WriteLn(System.RandSeed);
end.

program u1;
  uses u0;
end.
```

In diesem Beispiel soll die Unit u0 allein compiliert werden. Anschließend soll eine Compilierung von u1 zusammen mit CLXxx erfolgen (wobei 'xx' die Version bezeichnet). Das Problem tritt hier auf, weil Unit u0 unter der Voraussetzung compiliert wurde, dass sie nie auf die Daten eines Package zugreift.

```
(*$IMPORTEDDATA ON*)
unit u0;
interface
implementation
begin
  WriteLn(System.RandSeed);
end.

program u1;
  uses u0;
end.
```

Das Problem lässt sich am einfachsten durch eine erneute Compilierung der Fehler verursachenden Unit mit aktivierter Option \$IMPORTEDDATA lösen.

4.1.3.6.391 PACKED ist hier nicht erlaubt (E2006)

Das Schlüsselwort packed ist nur für Mengen-, Array-, Datensatz-, Objekt-, Klassen- und Dateitypen zulässig. Im Gegensatz zur 16-Bit-Version von Delphi wirkt sich packed auf das Layout von Datensatz-, Objekt- und Klassentypen aus.

```
program Produce;
```

```
type
  SmallReal = packed Real;
begin
end.
```

Packed kann nicht auf einen reellen Typ angewandt werden – wenn Sie Speicherplatz sparen möchten, müssen Sie den kleinsten reellen Typ, den Typ Single, verwenden.

```
program Solve;
type
  SmallReal = Single;
begin
end.
```

4.1.3.6.392 Parameterlose Konstruktoren sind in Record-Typen nicht zulässig (E2394)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.393 Nur Methoden von abgeleiteten Typen dürfen über Assemblierungsgrenzen hinweg auf das protected-Symbol [%s]%-s.%s zugreifen (E2363)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.394 Eigenschaftsdeklaration verweist auf private-Vor Fahr '%s.%s' (W1045)

Diese Warnung weist darauf hin, dass Ihr Quelltext nicht nach C++ portierbar ist. Dies ist für Komponentenentwickler wichtig, die Ihre Steuerelemente weitergeben möchten.

In Delphi können Sie in derselben Unit eine Basisklasse mit einem privaten Element und eine untergeordnete Klasse deklarieren, in der auf dieses Element verwiesen wird. In C++ ist diese Konstruktion nicht zulässig. Ändern Sie daher die untergeordnete Komponente so, dass der Verweis entweder auf ein protected-Element der Basisklasse oder auf ein protected-Element der untergeordneten Klasse erfolgt.

Hier ein Beispiel für Quelltext, der zu dieser Warnmeldung führt:

```
type
  TBase = class(...)
    private
      FFoo: Integer
    end;
  TChild=class(TBase)
    published
      property foo: Integer read FFoo write FFoo;
    end;
```

4.1.3.6.395 Das private-Symbol '%s' wurde deklariert, aber nie verwendet (H2219)

Das angegebene Symbol existiert zwar im private-Abschnitt einer Klasse, wird aber nie von der Klasse verwendet. Sie sparen Speicherplatz, wenn Sie nicht verwendete, als private deklarierte Felder aus der Klassendefinition löschen.

```
program Produce;
type
```

```

Base = class
private
  FVar : Integer;
  procedure Init;
end;

procedure Base.Init;
begin
end;

begin
end.

```

In diesem Beispiel wird eine Variable als private deklariert, aber nie verwendet. Dies führt zur Ausgabe einer Fehlermeldung durch den Compiler.

```

program Solve;
program Produce;
type
  Base = class
private
  FVar : Integer;
  procedure Init;
end;

procedure Base.Init;
begin
  FVar := 0;
end;

begin
end.

```

Für dieses Problem gibt es verschiedene Lösungen. Da es sich hier nicht um einen Fehler handelt, sind alle aufgeführten Lösungsvorschläge korrekt. Wenn das als private deklarierte Feld für eine spätere Verwendung vorgesehen ist, können Sie die Meldung einfach ignorieren. Ist die Variable wirklich überflüssig, kann sie problemlos entfernt werden. Es kann aber auch sein, dass die Verwendung der Variable im Quelltext einfach vergessen wurde. In diesem Fall fügen Sie den entsprechenden Programmtext hinzu.

4.1.3.6.396 PRIVATE oder PROTECTED erwartet (E2375)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.397 PROCEDURE, FUNCTION oder CONSTRUCTOR erwartet (E2357)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.398 PROCEDURE oder FUNCTION erwartet (E2122)

Diese Fehlermeldung wird von zwei unterschiedlichen Konstrukten ausgelöst. In beiden Fällen erwartet der Compiler das Schlüsselwort procedure oder function.

```

program Produce;

type
  Base = class
    class AProcedure; (*Fall 1*)
  end;

  class Base.AProcedure; (*Fall 2*)

```

```
begin
end;
```

```
begin
end.
```

In diesen beiden Fällen muss das Wort procedure auf das Schlüsselwort class folgen.

```
program Solve;
```

```
type
  Base = class
    class procedure AProcedure;
  end;

  class procedure Base.AProcedure;
begin
end;
```

```
begin
end.
```

Durch Hinzufügen von procedure tritt der Fehler nicht mehr auf.

4.1.3.6.399 Eigenschaftsdeklarationen sind im anonymen Record- oder lokalem Record-Typ nicht zulässig (E2434)

Record-Typen, die in lokalen Gültigkeitsbereichen oder in Variablen-deklarationen deklariert sind, dürfen nur Felddeklarationen enthalten. Für erweiterte Features in Record-Typen (wie z.B. Methoden, Eigenschaften und verschachtelte Typen) muss der Record-Typ explizit als globaler Typ definiert werden.

4.1.3.6.400 Dynamische oder Botschaftsmethoden sind hier nicht erlaubt (E2148)

Dynamische und Botschaftenmethoden können nicht als Zugriffsfunktionen für Eigenschaften benutzt werden.

```
program Produce;

type
  Base = class
    v : Integer;
    procedure SetV(x : Integer); dynamic;
    function GetV : Integer; message;
    property Velocity : Integer read GetV write v;
    property Value : Integer read v write SetV;
  end;

  procedure Base.SetV(x : Integer);
begin v := x;
end;

  function Base.GetV : Integer;
begin GetV := v;
end;

begin
end.
```

Sowohl Velocity als auch Value in diesem Beispiel sind fehlerhaft, da beiden Eigenschaften unzulässige Zugriffsfunktionen zugewiesen wurden.

```
program Solve;
```

```

type
  Base = class
    v : Integer;
    procedure SetV(x : Integer);
    function GetV : Integer;
    property Velocity : Integer read GetV write v;
    property Value : Integer read v write SetV;
  end;

procedure Base.SetV(x : Integer);
begin v := x;
end;

function Base.GetV : Integer;
begin GetV := v;
end;

begin
end.

```

4

Als Lösung wurden in diesem Beispiel die betreffenden Compileranweisungen aus den Deklarationen der Prozeduren entfernt; dies könnte allerdings in Ihrem speziellen Fall nicht von Nutzen sein. Eventuell müssen Sie die Logik Ihres Programms einer genauen Untersuchung unterziehen, um festzustellen, auf welche Weise Sie am besten Zugriffsfunktionen für die Eigenschaften zur Verfügung stellen können.

4.1.3.6.401 Auf Eigenschaft '%s' kann hier nicht zugegriffen werden (E2233)

Sie haben versucht, über einen Klassenreferenztyp auf eine Eigenschaft zuzugreifen. Der Zugriff auf die Felder oder Eigenschaften einer Klasse über eine Klassenreferenz ist aber nicht möglich.

```

program Produce;

type
  TBase = class
  public
    FX : Integer;
    property X : Integer read FX write FX;
  end;

TBaseClass = class of TBase;

var
  BaseRef : TBaseClass;
  x: Integer;

begin
  BaseRef := TBase;
  x := BaseRef.X;
end.

```

In diesem Beispiel führt der Versuch, auf die Eigenschaft X zuzugreifen, zur Ausgabe einer Fehlermeldung durch den Compiler.

```

program Solve;

type
  TBase = class
  public
    FX : Integer;
    property X : Integer read FX write FX;
  end;

TBaseClass = class of TBase;

```

```

var
  BaseRef : TBaseClass;
  x: Integer;

begin
  BaseRef := TBase;
end.

```

Das Problem lässt sich nur durch Entfernen des Fehler verursachenden Eigenschaftszugriffs aus dem Quelltext lösen. Wenn ein Zugriff auf die Eigenschaften oder Felder einer Klasse erforderlich ist, erzeugen Sie eine Instanzvariable für die betreffende Klasse und greifen über diese auf die Eigenschaft bzw. auf das Feld zu.

4.1.3.6.402 Das Eigenschaftsattribut 'label' darf nicht leer bleiben (E2275)

```

unit Problem;
interface
  type
    T0 = class
      f : integer;
      property g : integer read f write f label '';
    end;

implementation
begin
end.

```

Der Fehler wird ausgegeben, weil das Label-Attribut für 'g' ein leerer String ist.

```

unit Solve;
interface
  type
    T0 = class
      f : integer;
      property g : integer read f write f label 'LabelText';
    end;

implementation
begin
end.

```

Als Lösung wurde das Label-Attribut durch einen nicht leeren String ersetzt.

4.1.3.6.403 '%s' muss eine Eigenschaft oder ein Feld der Klasse '%s' referenzieren (E2292)

In der Attributdeklarationssyntax können Sie Werte, gefolgt von Name=Wert-Paaren an den Konstruktor der Attributklasse übergeben. Dabei ist 'Name' eine Eigenschaft oder Feld der Attributklasse.

4.1.3.6.404 Einer Nur-Lesen Eigenschaft kann kein Wert zugewiesen werden (E2129)

Sie versuchen, einer Eigenschaft ohne write-Klausel einen Wert zuzuweisen. Eine solche Eigenschaft wird jedoch als Nur-Lesen-Eigenschaft betrachtet.

```

program Produce;

type
  Base = class
    s: String;

```

```

    property Title: String read s;
  end;

var
  c : Base;

procedure DiddleTitle
begin
  if c.Title = '' then
    c.Title := 'Super Galactic Invaders with Turbo Gungla Sticks';

  (* weitere Aktionen mit c.Title*)
end;

begin
end.

```

Eigenschaften ohne write-Klausel sind Nur-Lesen-Eigenschaften. Da es nicht möglich ist, einer solchen Eigenschaft einen Wert zuzuweisen, löst die Zuweisung an c.Title eine Fehlermeldung des Compilers aus.

```

program Solve;

type
  Base = class
    s: String;

    property Title: String read s;
  end;

var
  c : Base;

procedure DiddleTitle
  var title : String;
begin
  title := c.Title;
  if Title = '' then
    Title := 'Super Galactic Invaders with Turbo Gungla Sticks';
  (* weitere Aktionen mit Title*)
end;

begin
end.

```

Wenn Sie über den Quelltext verfügen, besteht eine mögliche Lösung darin, für die Eigenschaft eine write-Klausel bereitzustellen. Diese Aktion stellt jedoch eine schwerwiegende Änderung der Basisklasse dar und sollte gut überlegt werden. Eine andere Möglichkeit ist, eine Zwischenvariable einzuführen, die den Wert der Nur-Lesen-Eigenschaft aufnimmt. Der obige Code zeigt dieses Verfahren.

4.1.3.6.405 Lesen einer Nur-Schreiben-Eigenschaft nicht möglich (E2130)

Sie versuchen, den Wert einer Eigenschaft ohne read-Klausel zu lesen. Eine solche Eigenschaft wird jedoch als Nur-Schreiben-Eigenschaft betrachtet.

```

program Produce;

type
  Base = class
    s: String;

    property Password : String write s;
  end;

var

```

```
c : Base;  
s: String;  
  
begin  
  s := c.Password;  
end.  
  
Da die Eigenschaft c.Password über keine read-Klausel verfügt, kann ihr Wert nicht gelesen werden.  
  
program Solve;  
  
type  
  Base = class  
    s: String;  
  
    property Password : String read s write s;  
  end;  
  
var  
  c : Base;  
  s: String;  
  
begin  
  s := c.Password;  
end.
```

Wenn Sie über den Quelltext verfügen, besteht eine einfache Lösung für dieses Problem darin, der Nur-Schreiben-Eigenschaft eine read-Klausel hinzuzufügen. Diese Lösung ist jedoch nicht immer ratsam, da sie zu Sicherheitslücken führen kann. Betrachten Sie beispielsweise eine Nur-Lesen-Eigenschaft mit dem Namen Passwort. Sicherlich ist es nicht sinnvoll, einem Programm zu erlauben, diese Klasse zum Lesen und Speichern von Passwörtern zu benutzen. Wenn eine Eigenschaft nicht gelesen werden kann, gibt es also möglicherweise einen guten Grund dafür.

4.1.3.6.406 Schreibweise von Eigenschaftszugriffsmethode %s.%s sollte %s.%s sein (x2367)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.407 Eigenschaftszugriffsmethode '%s' kann nicht erzeugt werden, weil '%s' bereits vorhanden ist (E2300)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.408 Geerbte Methoden können für den Zugriff auf Interface-Eigenschaften nicht verwendet werden (E2370)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.409 Eigenschaftszugriffsmethode '%s' sollte %s sein (H2369)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.410 Sichtbarkeit der Eigenschaftszugriffsmethode %s sollte mit der Eigenschaft %s.%s übereinstimmen (H2368)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.411 Redeklaration der Eigenschaft im OLE-Automatisierungsbereich nicht erlaubt (E2181)

Es ist nicht zulässig, die Sichtbarkeit einer Eigenschaft in einen Automatisierungsabschnitt zu stellen.

```
program Produce;

type
  Base = class
    v : Integer;
    s: String;
  protected
    property Name : String read s write s;
    property Value : Integer read v write v;
  end;

  Derived = class (Base)
  public
    property Name; (* Name durch Redeklaration öffentlich sichtbar machen *)
  automated
    property Value;
  end;

begin
end.
```

In diesem Beispiel wird Name aus privater Sichtbarkeit in Base mittels Neudeklaration zu öffentlicher Sichtbarkeit in Derived gebracht. Dieser Vorgang wird für Value wiederholt, führt jedoch zu einem Fehler.

```
program Solve;

type
  Base = class
    v : Integer;
    s: String;
  protected
    property Name : String read s write s;
    property Value : Integer read v write v;
  end;

  Derived = class (Base)
  public
    property Name; (* Name durch Redeklaration öffentlich sichtbar machen *)
    property Value;
  automated
  end;

begin
end.
```

Es ist nicht möglich, die Sichtbarkeit einer Eigenschaft für einen Automatisierungsabschnitt zu ändern. Daher liegt die Lösung dieses Problems darin, Eigenschaften von Basisklassen nicht in Automatisierungsabschnitten zu redeklarieren.

4.1.3.6.412 Das Überschreiben von Eigenschaften ist im Typ interface nicht erlaubt (E2206)

Eine in einer Basischnittstelle deklarierte Eigenschaft wurde in einer Schnittstellenerweiterung überschrieben.

```
program Produce;
type
  Base = interface
    function Reader : Integer;
    function Writer(a : Integer);
    property Value : Integer read Reader write Writer;
  end;

  Extension = interface (Base)
    function Reader2 : Integer;
    property Value Integer read Reader2;
  end;

begin
end.
```

In diesem Beispiel tritt dieser Fehler auf, weil Extension versucht, die Eigenschaft Value zu überschreiben.

```
program Solve;
type
  Base = interface
    function Reader : Integer;
    function Writer(a : Integer);
    property Value : Integer read Reader write Writer;
  end;

  Extension = interface (Base)
    function Reader2 : Integer;
    property Value2 Integer read Reader2;
  end;

begin
end.
```

Zur Lösung dieses Problems benennen Sie die betreffende Eigenschaft um. Sicherer ist es jedoch, die ursprüngliche Absicht festzustellen und den Systementwurf neu zu strukturieren.

4.1.3.6.413 Zugriff auf Eigenschaft muss über Instanzenfeld oder -methode erfolgen (E2356)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.414 Protected-Element '%s' ist in diesem Zusammenhang nicht verfügbar (E2389)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.415 Auf protected-Symbol %s.%s kann nicht zugegriffen werden (E2362)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.416 Der Zeigerausdruck benötigt kein Initialize/Finalize, aber möglicherweise einen ^-Operator (H2244)

Sie haben versucht, finalize auf einen Zeigertyp anzuwenden.

```
program Produce;

var
  str : String;
  pstr : PString;

begin
  str := 'Sharene';
  pstr := @str;
  Finalize(pstr); (*Hinweis: Versuchen Sie nicht, 'str' nachfolgend anzuwenden*)
end.
```

In diesem Beispiel wird der Zeiger pstr an die Prozedur Finalize übergeben. Dies erzeugt eine Meldung, da Finalize nicht auf Zeiger angewendet werden braucht.

```
program Solve;

var
  str : String;
  pstr : PString;

begin
  str := 'Sharene';
  pstr := @str;
  Finalize(pstr^); (*Hinweis: Versuchen Sie nicht, 'str' nachfolgend anzuwenden*)
end.
```

Die Lösung besteht darin, den Operator '^' zusammen mit dem übergebenen Zeiger zu verwenden.

4.1.3.6.417 Die Published-Real-Eigenschaft '%s' muss vom Typ Single, Real, Double oder Extended sein (E2186)

Sie haben versucht, eine Eigenschaft vom Typ Real zu veröffentlichen; dies ist nicht zulässig. Als published deklarierte Gleitkomma-Eigenschaften können Single, Double oder Extended sein.

```
program Produce;
type
  Base = class
    R : Real48;
  published
    property RVal : Real read R write R;
  end;
end.
```

Die published-Eigenschaft Real48 im obigen Programm muss entweder entfernt, in einen nichtöffentlichen Abschnitt gebracht oder in einen zulässigen Typ geändert werden.

```
program Produce;
type
  Base = class
    R : Single;
  published
    property RVal : Single read R write R;
  end;
end.
```

Mit dieser Lösung wird die Eigenschaft in einen real-Typ umgewandelt, mit dem Laufzeit-Typinformationen angezeigt werden.

4.1.3.6.418 Größe von Published-Menge '%s' ist >4 Byte (E2187)

Der Compiler lässt in einem published-Abschnitt keine Mengen zu, die größer als 32 Bit sind. Die Größe einer Menge in Byte kann folgendermaßen berechnet werden: High(setname) div 8 - Low(setname) div 8 + 1.

```
(*$TYPEINFO ON*)
program Produce;
type
  CharSet = set of Char;
  NamePlate = class
    Characters : CharSet;
  published
    property TooBig : CharSet read Characters write Characters ;
  end;

begin
end.

(*$TYPEINFO ON*)
program Solve;
type
  CharSet = set of 'A'..'Z';
  NamePlate = class
    Characters : CharSet;
  published
    property TooBig : CharSet read Characters write Characters ;
  end;

begin
end.
```

4.1.3.6.419 Auf private-Symbol %s.%s kann nicht zugegriffen werden (E2361)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.420 Klasse muss sealed sein, um einen private-Konstruktor ohne Typqualifizierer aufzurufen (E2390)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.421 Reihenfolge der Felder in der Record-Konstante unterscheidet sich von der Deklaration (E2083)

Diese Fehlermeldung wird angezeigt, wenn die Record-Felder in einer typisierten Konstante oder initialisierten Variable nicht in der Reihenfolge der Deklaration initialisiert werden.

```
program Produce;

type
  TPoint = record
    X, Y: Integer;
  end;

var
  Point : TPoint = (Y: 123; X: 456);

begin
```

```
end.
```

In diesem Beispiel wird Y vor X initialisiert. Dies entspricht jedoch nicht der Reihenfolge der Felder in der Deklaration.

```
program Solve;

type
  TPoint = record
    X, Y: Integer;
  end;

var
  Point : TPoint = (X: 456; Y: 123);

begin
end.
```

Initialisieren Sie Felder immer in der Reihenfolge der Deklaration.

4

4.1.3.6.422 Record-Typ zu groß: 1 MB überschritten (E2419)

Die Größe von Records ist entsprechend der .NET SDK-Dokumentation auf 1 MB beschränkt. Siehe Abschnitt II Metadata 21.8 ClassLayout: 0x0F

4.1.3.6.423 Rekursive Include-Datei %s (E2245)

Die Direktive \$I wurde zur rekursiven Einbindung einer anderen Datei verwendet. Sie müssen sicherstellen, dass alle eingebundenen Dateien keine Rekursionen enthalten.

4.1.3.6.424 Das Programm oder die Unit %s ruft sich selbst wieder auf (F2092)

Eine Unit hat versucht, sich selbst aufzurufen.

```
unit Produce;
interface
  uses Produce;
implementation

begin
end.
```

In diesem Beispiel wird in der uses-Klausel die Unit selbst referenziert. Der Compiler reagiert darauf mit der obigen Fehlermeldung.

```
unit Solve;
interface
implementation

begin
end.
```

Entfernen Sie die uses-Klausel, die den Fehler verursacht.

4.1.3.6.425 Package %s wird rekursiv benötigt (E2214)

Der Compiler hat beim Compilieren eines Package festgestellt, dass das Package sich selbst anfordert.

```
package Produce;
  requires Produce;
```

end.

Wenn ein Package sich unzulässigerweise selbst anfordert, tritt ein Fehler auf.

Entfernen Sie die rekursive Verwendung des Package.

4.1.3.6.426 Klassenmethoden in Record-Typen müssen static sein (E2398)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.427 Wiederhervorrufen einer Exception ist nur im Exception-Handler möglich (E2145)

Sie haben die Syntax der raise-Anweisung benutzt, die für die Neuauslösung einer Exception verwendet wird, aber der Compiler hat festgestellt, dass diese Neuauslösung außerhalb eines Exception-Behandlungsblockes erfolgt ist. Eine Einschränkung des aktuellen Mechanismus zur Exception-Behandlung unterbindet die Neuauslösung von Exceptions aus verschachtelten Exception-Behandlungsroutinen heraus.

```
program Produce;

procedure RaiseException;
begin
  raise;           (*Fall 1*)
  try
    raise;         (*Fall 2*)
  except
    try
      raise;       (*Fall 3*)
    except
    end;
    raise;
  end;
end;

begin
end.
```

Es gibt verschiedene Gründe, aus denen dieser Fehler auftreten kann. Erstens: Möglicherweise haben Sie eine Auslösung ohne Exception-Konstruktor außerhalb eines Exception-Handlers festgelegt. Zweitens: Sie haben eventuell versucht, eine Exception im try-Block eines Exception-Handlers neu auszulösen. Drittens: Vielleicht haben Sie versucht, die Exception in einem Exception-Handler neu auszulösen, der in einem anderen Exception-Handler verschachtelt ist.

```
program Solve;
  uses SysUtils;

procedure RaiseException;
begin
  raise Exception.Create('Fall 1');
  try
    raise Exception.Create('Fall 2');
  except
    try
      raise Exception.Create('Fall 3');
    except
    end;
    raise;
  end;
end;

begin
```

end.

Eine Lösung dieses Fehlers liegt darin, explizit eine neue Exception auszulösen; dies ist wahrscheinlich in Situationen wie "Fall 1" und "Fall 2" die Intention gewesen. In der Situation von "Fall 3" sollten Sie Ihren Quelltext untersuchen, um eine geeigneten Umweg zu finden, mit dem die gewünschten Ergebnisse erzielt werden können.

4.1.3.6.428 Borland.Delphi.Compiler.ResCvt.dll konnte nicht gefunden werden (E2377)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.429 Länge des Ressourcen-Strings überschreitet die Windows-Grenze von 4096 Zeichen (E2381)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.430 Ungültiger Funktionsrückgabetyp (E2024)

Dateitypen sind ungültige Funktionsergebnisse.

```
program Produce;

function OpenFile(Name: string): File;
begin
end;

begin
end.
```

Funktionen können keine Dateien liefern.

```
program Solve;

procedure OpenFile(Name: string; var F: File);
begin
end;

begin
end.
```

Die Datei kann als Variablenparameter "zurückgegeben" werden. Alternativ können Sie eine Datei dynamisch allozieren und einen Zeiger darauf zurückgeben.

4.1.3.6.431 Linker-Fehler: %s

Der Ressourcen-Linker (RLINK32) hat beim Bearbeiten einer Ressourcendatei einen Fehler entdeckt. Dafür kann es folgende Gründe geben:

- Sie haben einen nicht eindeutigen Ressourcennamen verwendet. Benennen Sie eine der Ressourcen um.
- Die Ressourcendatei ist beschädigt. Entfernen Sie die Datei, oder ersetzen Sie sie durch eine nicht beschädigte Version.
- Sie verwenden einen Ressourcentyp, der nicht unterstützt wird, z. B. eine 16-Bit-Ressource oder eine Formularschablone.
- Es gibt vielleicht ein Problem beim Konvertieren von 16-Bit-Symbolen nach 32 Bit.

4.1.3.6.432 Linker-Fehler: %s: %s

Der Ressourcen-Linker (RLINK32) hat beim Bearbeiten einer Ressourcendatei einen Fehler entdeckt. Eine zum Projekt gelinkte Ressource stimmt in ihrem Typ und Namen oder in ihrem Typ und ihrer Ressourcen-ID mit einer anderen, zum Projekt gelinkten Ressource überein. (Delphi gibt eine Warnung aus und ignoriert den Fehler. in Kylix verursachen nicht eindeutige Ressourcen einen Fehler.)

4.1.3.6.433 16-Bit-Segment in Objektdatei '%s' entdeckt (E2215)

In einer mit der Direktive \$L geladenen Objektdatei wurde ein 16-Bit-Segment gefunden.

Das Problem kann nur dadurch gelöst werden, dass Sie eine entsprechende Datei ohne 16-Bit-Segmentdefinition bereitstellen. Prüfen Sie in der Dokumentation zu dem Produkt, von dem die Objektdatei erstellt wurde, wie 16-Bit-Segmentdefinitionen in 32-Bit-Segmentdefinitionen umgewandelt werden können.

4.1.3.6.434 Segment/Offset-Paare werden in CodeGear 32-Bit-Pascal nicht unterstützt (E2091)

32-Bit-Code verwendet nicht mehr das Segment-/Offset-Adressierungsschema des 16-Bit-Codes.

In 16-Bit-Versionen von CodeGear Pascal wurden Segment-/Offset-Paare bei der Deklaration absoluter Variablen und als Argumente für die Standardfunktion Ptr verwendet.

In 32-Bit-Programmen für den Protected Mode sollten Sie keine absoluten Adressen verwenden. Rufen Sie stattdessen die entsprechenden Funktionen der Win32-API auf.

```
program Produce;

var
  VideoMode : Integer absolute $0040:$0049;

begin
  Writeln( Byte(Ptr($0040,$0049)^) );
end.
  program Solve;
(* Diese Version wird zwar kompiliert, ist jedoch nicht lauffähig. Verwenden Sie keine
absoluten Adressen! *)
var
  VideoMode : Integer absolute $0040*16+$0049;

begin
  Writeln( Byte(Ptr($0040*16+$0049)^) );
end.
```

4.1.3.6.435 ';' nicht erlaubt vor einem 'ELSE' (E2153)

Sie haben ein ";" unmittelbar vor einem else in einer if..else-Anweisung gesetzt. Der Grund für die Fehlermeldung ist, dass ";" als Anweisungs-Trennsymbol anstatt als Anweisungs-Endesymbol behandelt wird – if..else ist eine einzige Anweisung, und ein ";" kann nicht innerhalb dieser Anweisung stehen (gilt nicht für zusammengesetzte Anweisungen).

```
program Produce;

var
  b : Integer;

begin
  if b = 10 then
```

```

b := 0;
else
  b := 10;
end.

```

In Delphi ist es nicht zulässig, ein ";" unmittelbar vor eine else-Anweisung zu setzen. In diesem Quelltext wird daher ein Fehler gemeldet.

```

program Solve;

var
  b : Integer;

begin
  if b = 10 then
    b := 0
  else
    b := 10;

  if b = 10 then begin
    b := 0;
  end
  else begin
    b := 10;
  end;
end.

```

Für dieses Problem gibt es zwei einfache Lösungen. Die erste Lösung ist die Entfernung des betreffenden ";". Die zweite Lösung liegt darin, zusammengesetzte Anweisungen für alle Teile der if..else-Anweisung zu erstellen. Wenn \$HINTS aktiviert sind, erhalten Sie einen Hinweis darüber, dass der zu "b" zugewiesene Wert niemals benutzt wird.

4.1.3.6.436 Methoden zum Setzen von Eigenschaften können keine var-Parameter übernehmen (E2282)

Diese Meldung wird angezeigt, wenn Sie in der Schreibmethode einer Eigenschaft einen var-Parameter angeben. Parameter von Schreibmethoden dürfen keine var- oder out-Parameter sein.

4.1.3.6.437 Mengen (Sets) dürfen nur maximal 256 Elemente besitzen (E2028)

Diese Fehlermeldung erscheint, wenn Sie versuchen, einen Mengentyp mit mehr als 256 Elementen zu deklarieren. Genauer gesagt müssen die Ober- und die Untergrenze des Basistyps innerhalb von 0 und 255 liegen.

```

program Produce;
type
  BigSet = set of 1..256;  (*<-- Hier wird die Fehlermeldung ausgegeben*)
begin
end.

```

In diesem Beispiel hat BigSet wirklich nur 256 Elemente, aber auch dies ist unzulässig.

```

program Solve;
type
  BigSet = set of 0..255;
begin
end.

```

Sie müssen darauf achten, dass die Ober- und die Untergrenze innerhalb von 0 und 255 liegen.

4.1.3.6.438 Standard-Funktion Slice nur als Argument für offene Arrays erlaubt (E2193)

Es wurde versucht, ein Array-Slice an ein Array mit fester Größe zu übergeben. Array-Slices können nur als Parameter für offene Arrays übergeben werden.

```
program Produce;

type
  IntegerArray = array [1..10] OF Integer;

var
  SliceMe : array [1..200] OF Integer;

procedure TakesArray(x : IntegerArray);
begin
end;

begin TakesArray(SLICE(SliceMe, 5));
end.
```

In diesem Beispiel wird der Fehler ausgelöst, weil TakesArray ein Array mit fester Größe erwartet.

```
program Solve;

type
  IntegerArray = array [1..10] OF Integer;

var
  SliceMe : array [1..200] OF Integer;

procedure TakesArray(x : array of Integer);
begin
end;

begin TakesArray(SLICE(SliceMe, 5));
end.
```

In diesem Beispiel wird der Fehler dagegen nicht ausgelöst, weil TakesArray ein offenes Array als Parameter annimmt.

4.1.3.6.439 Slice-Standardfunktion für VAR- und OUT-Argument nicht zulässig (E2454)

Ein Teil eines Array kann nicht zurückgeschrieben werden, daher kann mit der Slice-Standardfunktion kein **var**- oder **out**-Argument übergeben werden. Wenn Sie das Array modifizieren müssen, übergeben Sie entweder das gesamte Array oder verwenden Sie eine Array-Variable, die den gewünschten Teil des gesamten Array aufnimmt.

4.1.3.6.440 \$EXTERNALSYM und \$NODEFINE sind für '%s' nicht erlaubt; nur globale Symbole (E2240)

Die Direktiven **\$EXTERNALSYM** und **\$NODEFINE** können nur auf globale Symbole angewendet werden.

4.1.3.6.441 Stringkonstante abgeschnitten damit sie in STRING[%ld] passt (W1014)

Eine Stringkonstante wird einer Variablen zugewiesen, die nicht groß genug ist, um den gesamten String aufnehmen zu können.

Der Compiler gibt eine Warnung aus, dass das Literal entsprechend der Variablengröße abgeschnitten wurde.

```
program Produce;
(*$WARNINGS ON*)

const
  Title = 'Super Galactic Invaders with Turbo Gunpla Sticks';
  Subtitle = 'Copyright (c) 2002 by Frank Borland';

type
  TitleString = String[25];
  SubtitleString = String[18];

var
  ProgramTitle : TitleString;
  ProgramSubtitle : SubtitleString;

begin
  ProgramTitle := Title;
  ProgramSubtitle := Subtitle;
end.
```

Die beiden Stringkonstanten werden Variablen zugewiesen, die zu kurz sind, um den gesamten String aufnehmen zu können. Der Compiler schneidet die Strings ab und führt die Zuweisung durch.

```
program Solve;
(*$WARNINGS ON*)

const
  Title = 'Super Galactic Invaders with Turbo Gunpla Sticks';
  Subtitle = 'Copyright (c) 2002';

type
  TitleString = String[55];
  SubtitleString = String[18];

var
  ProgramTitle : TitleString;
  ProgramSubtitle : SubtitleString;

begin
  ProgramTitle := Title;
  ProgramSubtitle := Subtitle;
end.
```

Für dieses Problem gibt es zwei mögliche Lösungen, die beide in diesem Beispiel gezeigt werden. Die erste Lösung ist die Vergrößerung der Variablen, sodass sie den gesamten String aufnehmen kann. Die zweite Möglichkeit liegt darin, die Größe des Strings zu verringern, damit er von der Variablen in ihrer deklarierten Größe aufgenommen werden kann.

4.1.3.6.442 String-Element kann nicht an var-Parameter übergeben werden (E2354)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.443 String-Literale können maximal 255 Elemente besitzen (E2056)

Diese Fehlermeldung wird angezeigt, wenn Sie einen String-Typ mit mehr als 255 Elementen deklarieren, einen literalen String mit mehr als 255 Zeichen einer Variable des Typs ShortString zuweisen oder in einen String mehr als 255 Zeichen aufnehmen.

Beachten Sie, dass Sie lange Strings über mehrere Zeilen hinweg erstellen können, indem Sie die einzelnen Zeichenfolgen mit

dem Operator + verketten.

```
program Produce;
var
  LongString : string[256];  (**<-- Hier die Fehlermeldung*)
begin
end.
```

Im obigen Beispiel ist der String um ein Zeichen zu lang.

```
program Solve;
var
  LongString : AnsiString;
begin
end.
```

Die beste Lösung ist, die neuen langen Strings zu verwenden. Sie brauchen dann keine Längenbeschränkungen mehr zu beachten.

4.1.3.6.444 Starker Namensschlüssel kann nicht aus Assemblierung %s extrahiert werden (E2408)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.445 Zweifelhafte Typumwandlung von %s in %s (W1044)

Diese Warnung wird bei Typumwandlungen wie PWideChar(String) oder PChar(WideString) angezeigt, in denen unterschiedliche String-Typen ohne Zeichenkonvertierung umgewandelt werden.

4.1.3.6.446 Der reservierte Unit-Name '%s' kann nicht verwendet werden (E2272)

Es wurde versucht, eine benutzerdefinierte Unit mit einem reservierten Unit-Namen, wie z. B. System, zu benennen.

Die folgenden Namen sind für den Compiler reserviert:

- System
- SysInit

```
unit System;
interface
implementation
begin
end.
```

In diesem Beispiel wird eine Unit mit einem Namen deklariert, der für die Verwendung durch den Compiler reserviert ist.

```
unit MySystem;
interface
implementation
begin
end.
```

Die einzige Lösung besteht darin, einen anderen Namen für die Unit zu wählen.

4.1.3.6.447 Ausdruck zu komplex (E2156)

Der Compiler ist in Ihrem Quelltext auf einen Ausdruck gestoßen, der für eine Compilierung zu komplex ist.

Reduzieren Sie die Komplexität des Ausdrucks, indem Sie temporäre Variablen einführen.

4.1.3.6.448 Zu viele lokale Konstanten. Verwenden Sie kürzere Prozeduren (E2283)

Eine oder mehrere Ihrer Prozeduren enthält so viele String-Konstanten, dass die interne Speichergrenze des Compilers überschritten wurde. Dieses Problem kann in automatisch generiertem Code auftreten. Kürzen Sie die betreffenden Prozeduren, oder deklarieren Sie Konstanten anstelle der vielen Literale.

4.1.3.6.449 Zu viele bedingte Symbole (E2163)

Sie haben den für bedingte Symbole (einschließlich Konfigurationsdateien) zugewiesenen Speicherplatz überschritten, der in der Befehlszeile definiert wurde. Für alle bedingten Symbole zusammen sind 256 Byte zugewiesen. Jedes bedingte Symbol belegt bei der Speicherung im Bereich für bedingte Symbole 1 zusätzliches Byte.

Die einzige Lösung liegt darin, die Anzahl der bedingten Compilierungssymbole in der Befehlszeile (oder in den Konfigurationsdateien) zu verringern.

4.1.3.6.450 Die Compilierung wurde wegen zu vieler Fehler abgebrochen (E2226)

Der Compiler hat die maximale Anzahl der Fehler überschritten, die bei einer Compilierung auftreten dürfen.

Sie müssen einige der aufgetretenen Fehler korrigieren und das Projekt erneut compilieren.

4.1.3.6.451 Zu viele Parameter (E2034)

Diese Fehlermeldung tritt auf, wenn beim Aufruf einer Prozedur oder einer Funktion mehr Parameter angegeben werden als von der Prozedur oder Funktion festgelegt.

Außerdem tritt diese Fehlermeldung auf, wenn ein OLE-Automatisierungsauftrag zu viele (mehr als 255) oder zu viele benannte Parameter aufweist.

```
program Produce;

function Max(A,B: Integer): Integer;
begin
  if A > B then Max := A else Max := B
end;

begin
  Writeln( Max(1,2,3) );    (*<-- Hier die Fehlermeldung*)
end.
```

Es wäre praktischer, wenn Max drei Parameter annehmen würde ...

```
program Solve;

function Max(const A: array of Integer): Integer;
var
  I: Integer;
begin
  Result := Low(Integer);
  for I := 0 to High(A) do
    if Result < A[I] then
      Result := A[I];
```

```

end;

begin
  Writeln( Max([1,2,3]) );
end.

```

Normalerweise würden Sie den Aufruf ändern, um die korrekte Anzahl von Parametern angeben zu können. Hier haben wir uns dazu entschieden, Ihnen zu zeigen, wie Max mit einer unbegrenzten Zahl von Parametern implementiert werden kann. Beachten Sie, dass der Aufruf von Max nun auf etwas andere Weise erfolgen muss.

4.1.3.6.452 Typisierte Konstante '%s' als Var-Parameter übergegeben (W1016)

Dies ist eine reservierte Fehlermeldung.

4.1.3.6.453 PUBLISHED verursachte, dass RTTI (\$M+) zu Typ '%s' hinzugefügt wurde (W1055)

Sie haben einen 'PUBLISHED'-Abschnitt zu einer Klasse hinzugefügt, die nicht mit der Option {\$M+}/{\$TYPEINFO ON} kompiliert wurde, oder ohne Ableitung von einer Klasse, die mit der Option {\$M+}/{\$TYPEINFO ON} kompiliert wurde.

Die Standardprozedur TypeInfo erwartet einen Typbezeichner als Parameter. Im obigen Beispiel stellt NotType keinen Typbezeichner dar.

Um diesen Fehler zu vermeiden, stellen Sie sicher, dass Sie nur mit der Option {\$M+}/{\$TYPEINFO ON} kompilieren, oder von einer Klasse ableiten, die mit der Option {\$M+}/{\$TYPEINFO ON} kompiliert wurde.

4.1.3.6.454 Standardfunktion TYPEINFO erwartet einen Typbezeichner (E2133)

Sie versuchen, Typinformationen für einen Bezeichner abzurufen, der keinen Typ darstellt.

```

program Produce;

var
  p : Pointer;

procedure NotType;
begin
end;

begin
  p := TypeInfo(NotType);
end.

```

Die Standardprozedur TypeInfo erwartet einen Typbezeichner als Parameter. Im obigen Beispiel stellt NotType keinen Typbezeichner dar.

```

program Solve;

type
  Base = class
end;

var
  p : Pointer;

begin
  p := TypeInfo(Base);
end.

```

Stellen Sie sicher, dass der Parameter für TypeInfo ein Typbezeichner ist.

4.1.3.6.455 Typdeklarationen sind im anonymen Record- oder lokalem Record-Typ nicht zulässig (E2436)

Record-Typen, die in lokalen Gültigkeitsbereichen oder in Variablendeklarationen deklariert sind, dürfen nur Felddeklarationen enthalten. Für erweiterte Features in Record-Typen (wie z.B. Methoden, Eigenschaften und verschachtelte Typen) muss der Record-Typ explizit als globaler Typ definiert werden.

4.1.3.6.456 '%s' ist kein Typenbezeichner (E2005)

Diese Fehlermeldung tritt auf, wenn der Compiler die Bezeichnung eines Typs erwartet hat, aber die gefundene Bezeichnung keinen Typ bezeichnet.

```
program Produce;
type
  TMyClass = class
    Field: Integer;
  end;
var
  MyClass : TMyClass;

procedure Proc(C: MyClass);           (*<-- Hier die Fehlermeldung*)
begin
end;

begin
end.
```

In diesem Beispiel wird als Typ des Argumentes unzulässigerweise die Bezeichnung der Variablen anstatt der Bezeichnung des Typs verwendet.

```
program Solve;
type
  TMyClass = class
    Field: Integer;
  end;
var
  MyClass : TMyClass;

procedure Proc(C: TMyClass);
begin
end;

begin
end.
```

Achten Sie darauf, dass der betreffende Bezeichner wirklich ein Typ ist – möglicherweise ist er falsch geschrieben, oder ein anderer Bezeichner mit derselben Bezeichnung verbirgt den Bezeichner, auf den Sie sich beziehen wollten.

4.1.3.6.457 Der Ausdruck benötigt kein Initialize/Finalize (x2243)

Sie haben versucht, die Standardprozedur Finalize auf einen Typ anzuwenden, der dies nicht erfordert.

```
program Produce;

var
  ch : Char;

begin
```

```
    Finalize(ch);
end.
```

In diesem Beispiel benötigt der Delphi-Typ Char keine finalize-Anweisung.

In der Regel reicht es aus, den Aufruf von Finalize zu entfernen.

4.1.3.6.458 Datentyp zu groß: 2 GB überschritten (E2100)

Sie haben einen Datentyp angegeben, der zu groß ist, um vom Compiler dargestellt zu werden. Dieser Fehler tritt bei Datentypen von mehr als 2 GB Größe auf. Verringern Sie die Größe in der Typbeschreibung.

```
program Produce;

type
  EnormousArray = array [0..MaxLongint] OF Longint;
  BigRecord = record
    points : array [1..10000] of Extended;
  end;

var
  data : array [0..500000] of BigRecord;

begin
end.
```

Der Grund für die Fehlermeldung ist in diesem Beispiel offensichtlich.

```
program Solve;
type
  EnormousArray = array [0..MaxLongint DIV 8] OF Longint;

  DataPoints = ^DataPointDesc;
  DataPointDesc = array [1..10000] of Extended;
  BigRecord = record
    points : DataPoints;
  end;

var
  data : array [0..500000] OF BigRecord;

begin
end.
```

Stellen Sie sicher, dass die Größe Ihrer Datentypen 2 GB nicht überschreitet. Notfalls müssen Sie Ihre Daten wie mit der BigRecord-Deklaration umstrukturieren.

4.1.3.6.459 Größe des Datentyps ist null (E2101)

Record-Typen müssen mindestens ein Instanzendatenfeld enthalten. Records mit der Größe null sind in .NET nicht zulässig.

4.1.3.6.460 Eigenschaft '%s' existiert nicht in Basisklasse (E2147)

Der Compiler nimmt an, dass Sie versuchen, eine Eigenschaft auf eine andere, höhere Sichtbarkeitsebene in einer abgeleiteten Klasse zu stellen, aber diese Klasse ist in der Basisklasse nicht vorhanden.

```
program Produce;

type
  Base = class
  private
```

```

a: Integer;
property BaseProp : integer read a write a;
end;

Derived = class (Base)
  ch : Char;
  property Alpha read ch write ch; (*Fall 1*)
  property BesaProp; (*Fall 2*)
end;

begin
end.

```

Für diesen Fehler gibt es zwei Hauptursachen. Zum einen tritt er durch Festlegung einer neuen Eigenschaft ohne Festlegung eines Typs auf; dies wird in der Regel nicht als Verschiebung auf eine andere Sichtbarkeitsebene angesehen. Die zweite Ursache liegt in der Festlegung einer Eigenschaft, die sich in der Basisklasse befinden sollte, aber nicht vom Compiler gefunden werden kann; am häufigsten ist ein Schreibfehler hierfür verantwortlich (z. B. in "BesaProp"). Bei der zweiten Ursache gibt der Compiler außerdem Fehlermeldungen aus, dass eine Lese- oder Schreibanweisung erwartet wird.

```

program Solve;

type
  Base = class
  private
    a: Integer;
    property BaseProp : integer read a write a;
  end;

  Derived = class (Base)
    ch : Char;
  public
    property Alpha : Char read ch write ch; (*Fall 1*)
    property BaseProp; (*Fall 2*)
  end;

begin
end.

```

Die Lösung im ersten Fall ist die Angabe eines Typs für die Eigenschaft. Im zweiten Fall liegt die Lösung darin, die Schreibweise der Bezeichnung der Eigenschaft zu überprüfen.

4.1.3.6.461 Unicode-Zeichen sind in published Symbolen nicht zulässig (E2452)

Das VCL-RTTI-Subsystem (Run-Time Type Information) und das Streaming der DFM-Dateien erfordern, dass **published**-Symbole Nicht-Unicode (ANSI) Zeichen sind. Überlegen Sie, ob dieses Symbol **published** sein muss und verwenden Sie dann ANSI-Zeichen anstelle von Unicode.

4.1.3.6.462 Fehler beim Konvertieren der Unicode-Zeichen in den länderspezifischen Zeichensatz. String wurde abgeschnitten. Ist die Umgebungsvariable LANG korrekt gesetzt? (W1041)

Diese Meldung wird angezeigt, wenn Sie einen Unicode-String in Ihren lokalen Zeichensatz konvertieren, und der String Zeichen enthält, die im aktuellen Gebietsschema nicht zulässig sind. Dies ist beispielsweise beim Konvertieren von WideString in AnsiString oder beim Versuch, japanische Zeichen in einem englischen Gebietsschema anzuzeigen, der Fall.

4.1.3.6.463 Unit '%s' wird abgelehnt (W1006)

Die Unit ist veraltet und nur aus Gründen der Abwärtskompatibilität vorhanden.

Die Unit ist (mit der Hinweisdirektive **deprecated**) als nicht mehr aktuell gekennzeichnet und nur aus Kompatibilitätsgründen vorhanden. Verwenden Sie möglichst eine andere Unit in Ihrem Quelltext.

Mit der Compiler-Direktive **\$WARN UNIT_DEPRECATED ON/OFF** können alle entsprechenden Warnungen für Units aktiviert oder deaktiviert werden.

4.1.3.6.464 Unit '%s' ist experimental (W1007)

Eine "experimental" Direktive wurde für einen Bezeichner verwendet. "Experimental" gibt an, dass eine Klasse oder Unit unvollständig oder nicht ausreichend getestet ist.

4.1.3.6.465 Falsches Unit-Format: '%s' (F2048)

Dieser Fehler tritt auf, wenn eine compilierte Unit-Datei (.dcu) ein falsches Format hat.

Die Datei ist höchstwahrscheinlich beschädigt. Compilieren Sie Datei erneut (sofern Sie Zugriff auf den Quelltext haben). Falls das Problem bestehen bleibt, muss Delphi unter Umständen neu installiert werden.

4.1.3.6.466 System.Runtime.CompilerServices.RunClassConstructor nicht gefunden. Unit-Initialisierungsreihenfolge entspricht nicht der Reihenfolge in der uses-Klausel (W1052)

Diese Warnung weist darauf hin, dass die in Delphi definierte Initialisierungsreihenfolge, die durch die Reihenfolge der Units in der **uses**-Klausel festgelegt ist, nicht garantiert ist.

Mit der Funktion RunClassConstructor werden die **initialization**-Abschnitte der Units, die von der aktuellen Unit verwendet werden, in der Reihenfolge ausgeführt, die in der **uses**-Klausel der aktuellen Unit festgelegt ist. Diese Warnung wird ausgegeben, wenn der Compiler diese Funktion in dem .NET-Framework finden kann. Wenn Sie z.B. mit dem .NET Compact Framework linken, das RunClassConstructor nicht implementiert, tritt diese Warnung auf.

4.1.3.6.467 Unit '%s' ist bibliotheksspezifisch (W1004)

Die gesamte Unit ist (mit der Hinweisdirektive **library**) als eine gekennzeichnet, die nicht in allen Bibliotheken zur Verfügung steht. Wenn Sie andere Bibliotheken verwenden, kann dies zu Problemen führen.

Mit der Compiler-Direktive **\$WARN UNIT_LIBRARY ON/OFF** können alle entsprechenden Warnungen für Units aktiviert oder deaktiviert werden.

4.1.3.6.468 Unit-Bezeichner '%s' stimmt mit dem Dateinamen nicht überein (E1038)

Da bei dem Unit-Namen in der obersten Unit zwischen Groß- und Kleinschreibung unterschieden wird, muss die Schreibweise genau übereinstimmen. Beim Unit-Namen findet diese Unterscheidung nur in der Unit-Deklaration statt.

4.1.3.6.469 Unit '%s' ist plattformspezifisch (W1005)

Die gesamte Unit ist (mit der Hinweisdirektive **platform**) als eine gekennzeichnet, die Inhalte enthält, die nicht auf allen Plattformen verfügbar sind. Wenn Sie plattformübergreifende Anwendungen erstellen, kann dies zu Problemen führen. Beispielsweise wird **platform** bei Units mit Objekten angegeben, die in **OleAuto** definiert sind.

Mit der Compiler-Direktive **\$WARN UNIT_PLATFORM ON/OFF** können alle derartigen Warnungen für Units aktiviert oder deaktiviert werden.

4.1.3.6.470 Unbekannte Direktive: '%s' (E2070)

Diese Fehlermeldung wird angezeigt, wenn der Compiler in einer Prozedur- oder Funktionsdeklaration auf eine unbekannte Direktive trifft.

Die Anweisung wurde wahrscheinlich falsch geschrieben, oder es wurde ein Semikolon vergessen.

```
program Produce;

procedure P; stcall;
begin
end;

procedure Q forward;

function GetLastError: Integer external 'kernel32.dll';

begin
end.
```

In der Deklaration von **P** wurde die Aufrufkonvention **stdcall** falsch geschrieben. In den Deklarationen von **Q** und **GetLastError** fehlt ein Semikolon.

```
program Solve;

procedure P; stdcall;
begin
end;

procedure Q; forward;

function GetLastError: Integer; external 'kernel32.dll';

begin
end.
```

Vergewissern Sie sich, dass Sie die Anweisungen richtig geschrieben und alle benötigten Semikolons eingegeben haben.

4.1.3.6.471 Linker-Fehler bei der Ausgabe von Metadaten (E2328)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.472 Unbekanntes Ressourcenformat '%s' (E2400)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.473 Abschnitt '%s' in Objektdatei '%s' kann nicht behandelt werden (E2216)

Sie versuchen, mit der Compiler-Direktive \$L Objektmodule zu Ihrem Programm zu linken. Der Compiler kann die Objektdatei jedoch nicht verarbeiten; sie ist zu komplex. Die Einbindung von C++ Objektdateien wird beispielsweise nicht unterstützt.

4.1.3.6.474 Unbekannten Elementtyp beim Importieren der Signatur von %s.%s gefunden (E2405)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.475 Feld-Offset kann für Variant-Record nicht festgestellt werden, weil der vorherige Feldtyp ein Record-Typ unbekannter Größe ist (E2417)

Private Typen in einer Assemblierung werden nicht importiert; sie erhalten die Kennzeichnung: nicht zuverlässige Größe. Wenn ein Record mit mindestens einem Private-Feld deklariert wird oder der Record ein Feld hat, dessen Typgröße nicht zuverlässig ist, tritt dieser Fehler auf.

4.1.3.6.476 Unbenannte Argumente müssen benannten Argumenten in der OLE-Automatisierung vorangestellt werden (E2166)

Sie haben versucht, unbenannte Parameter nach benannten OLE-Automatisierungsparametern zu setzen.

```
program Produce;  
  
  var  
    ole : variant;  
  
begin ole.dispatch(filename:='FrogEggs', 'Tapioca');  
end.
```

Der benannte Parameter FileName muss in dieser OLE-Verteilung nach dem unbenannten Parameter stehen.

```
program Solve;  
  
  var  
    ole : variant;  
  
begin ole.dispatch('Tapioca', filename:='FrogEggs');  
end.
```

Die Lösung, die Parameter zu vertauschen, ist am einfachsten, kann sich jedoch für Ihren speziellen Fall als ungeeignet erweisen. Eine andere Möglichkeit liegt darin, dem unbenannten Parameter eine Bezeichnung zu geben.

4.1.3.6.477 Nicht aufgelöstes benutzerdefiniertes Attribut: %s (E2289)

Nach einer benutzerdefinierten Attributdeklaration folgte keine Symboldeklaration, wie z.B. eine Typ-, Variablen-, Methoden- oder Parameterdeklaration.

4.1.3.6.478 Unsicherer Code ist nur in unsicheren Prozeduren zulässig (E2396)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.479 Unsichere Prozeduren sind nur bei der Compilierung mit {\$UNSAFECODE ON} zulässig (E2395)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.480 Unsichere Zeiger sind nur bei der Compilierung mit {\$UNSAFECODE ON} zulässig (E2397)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.481 Unsichere Zeigervariablen, -parameter oder -konstanten sind nur in unsicheren Prozeduren zulässig (E2410)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.482 Zweifelhafte Typumwandlung von '%s' in '%s' (W1048)

Sie haben einen Datentyp verwendet oder einen Vorgang ausgeführt, der möglicherweise Speicher überschreibt. In einer gesicherten Ausführungsumgebung wie beispielsweise .NET wird solcher Code als unsicher und potentiell gefährlich betrachtet.

4.1.3.6.483 Unsafe-Code: '%s' (W1047)

Sie haben einen Datentyp verwendet oder einen Vorgang ausgeführt, der möglicherweise Speicher überschreibt. In einer gesicherten Ausführungsumgebung wie beispielsweise .NET wird solcher Code als unsicher und potentiell gefährlich betrachtet.

4.1.3.6.484 EXPORTS-Abschnitte sind nur bei der Compilierung mit {\$UNSAFECODE ON} zulässig (E2406)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.485 Unsicherer Typ: '%s%s%s' (W1046)

Sie haben einen Datentyp verwendet oder einen Vorgang ausgeführt, der möglicherweise Speicher überschreibt. In einer gesicherten Ausführungsumgebung wie beispielsweise .NET wird solcher Code als unsicher und potentiell gefährlich betrachtet.

4.1.3.6.486 Sprach-Feature wird nicht unterstützt: '%s' (x1025)

Beim Versuch, eine Delphi-Unit in eine C++ Header-Datei zu übersetzen, wurden in der Unit Sprachelemente entdeckt, die nicht unterstützt werden.

Damit die Unit übersetzt werden kann, muss das fehlerhafte Konstrukt aus dem interface-Abschnitt entfernt werden.

4.1.3.6.487 Unerwartetes Dateiende im in Zeile %Id beginnenden Kommentar (E2057)

Dieser Fehler tritt auf, wenn Sie einen Kommentar öffnen, aber nicht schließen.

Beachten Sie, dass Sie mit "{" beginnende Kommentare mit dem Zeichen "}" und mit "(*" beginnende Kommentare mit der Zeichenfolge ")" abschließen müssen.

```
program Produce;
(*Hier ein Kommentar, der nicht geschlossen wurde
begin
end.
```

Hier wurde vergessen, den Kommentar abzuschließen.

```
program Solve;
(*Hier ein geschlossener Kommentar *)
begin
end.
```

Sie brauchen nur den Kommentar abzuschließen.

4.1.3.6.488 Nicht abgeschlossene bedingte Direktive (E2280)

Für jedes {\$IFxxx} muss das zugehörige {\$ENDIF} oder {\$IFEND} in derselben Quelltextdatei definiert sein. Diese Meldung zeigt an, dass zu wenige Abschlussdirektiven vorkommen.

Die Fehlermeldung wird in der Zeile mit dem letzten \$IF/\$IFDEF/usw. ohne entsprechendes \$ENDIF/\$IFEND angezeigt. Sie können an dieser Stelle mit der Suche nach dem Problem beginnen.

4.1.3.6.489 Nicht abgeschlossener String (E2052)

Der Compiler konnte das abschließende Anführungszeichen eines Strings nicht finden.

Beachten Sie, dass Strings nicht in der nächsten Zeile fortgesetzt werden können. Sie können aber zwei Strings in aufeinander folgenden Zeilen mit dem Operator + verketten.

```
program Produce;

begin
  Writeln('Hello world!'); (*-- Hier die Fehlermeldung --*)
end.
```

Hier wurde nur vergessen, den String mit einem Anführungszeichen abzuschließen.

```
program Solve;

begin
  Writeln('Hello world! ');
end.
```

Sie brauchen nur das fehlende Anführungszeichen einzugeben.

4.1.3.6.490 Variable '%s' wurde deklariert, aber in '%s' nicht verwendet (H2164)

Sie haben in einer Prozedur eine Variable deklariert, verwenden sie jedoch nicht.

```
program Produce;
(*$HINTS ON*)
```

```

procedure Local;
  var i: Integer;
begin
end;

begin
end.
  program Solve;

(*$HINTS ON*)

procedure Local;
begin
end;

begin
end.

```

Eine einfache Lösung liegt darin, alle unbenutzten Variablen aus Ihren Prozeduren zu entfernen. Unbenutzte Variablen können allerdings auch auf einen Fehler bei der Implementierung Ihres Algorithmus hindeuten.

4.1.3.6.491 Package-Unit '%s' kann nicht in den Klauseln contains oder uses erscheinen (E2212)

Die in der Fehlermeldung angegebene Unit ist eine Package-Unit, die als solche nicht in Ihrem Projekt vorhanden sein darf. Möglicherweise wurde einer Delphi- und einer Package-Unit derselbe Name gegeben. Der Compiler findet in seinem Suchpfad zuerst die Package-Unit, bevor eine gleichnamige Delphi-Datei lokalisiert werden kann. Packages dürfen nicht über die Angabe einer Package-Unit in der uses-Klausel in ein Projekt eingefügt werden.

4.1.3.6.492 Verwendete Unit '%s' kann nicht compiliert werden (F2063)

Dieser schwerwiegende Fehler tritt auf, wenn eine Unit nicht compiliert werden kann, die von einer anderen Unit verwendet wird. Der Compiler bricht in diesem Fall die Bearbeitung der abhängigen Unit ab, da wahrscheinlich sehr viele Folgefehler auftreten.

4.1.3.6.493 Unterbrechung durch Benutzer - Compilierung abgebrochen (E2090)

Diese Meldung wird momentan nicht verwendet.

4.1.3.6.494 Compilierung durch Anwender abgebrochen (E2165)

Sie haben während einer Compilierung die Tastenkombination Strg+Pause betätigt.

4.1.3.6.495 Variable '%s' ist möglicherweise nicht initialisiert worden (W1036)

Diese Warnung wird angezeigt, wenn einer Variablen nicht in jedem Codepfad, der zu der Stelle führt, an der sie verwendet wird, ein Wert zugewiesen wird.

```

program Produce;
(*$WARNINGS ON*)
var
  B : Boolean;
  C: (Red,Green,Blue);

```

```

procedure Simple;
var
  I: Integer;
begin
  Writeln(I);          (*<-- Hier die Warnmeldung*)
end;

procedure IfStatement;
var
  I: Integer;
begin
  if B then
    I := 42;
  Writeln(I);          (*<-- Hier die Warnmeldung*)
end;

procedure CaseStatement;
var
  I: Integer;
begin
  case C of
    Red..Blue: I := 42;
  end;
  Writeln(I);          (*<-- Hier die Warnmeldung*)
end;

procedure TryStatement;
var
  I: Integer;
begin
  try
    I := 42;
  except
    Writeln('Should not get here!');
  end;
  Writeln(I);          (*<-- Hier die Warnmeldung*)
end;

begin
  B := False;
end.

```

Sie müssen in einer if-Anweisung sicherstellen, dass die Variable in beiden Verzweigungen zugewiesen wird. Bei einer case-Anweisung muss ein else-Abschnitt hinzugefügt werden, damit die Variable in jedem möglichen Fall einen Wert erhält. Bei einem try-except-Konstrukt geht der Compiler davon aus, dass die Anweisungen im try-Abschnitt auch dann nicht ausgeführt werden, wenn Sie sich am Anfang des Abschnitts befinden und so einfach sind, dass eigentlich keine Exception auftreten kann.

```

program Solve;
(*$WARNINGS ON*)
var
  B : Boolean;
  C: (Red,Green,Blue);

procedure Simple;
var
  I: Integer;
begin
  I := 42;
  Writeln(I);
end;

procedure IfStatement;
var
  I: Integer;
begin
  if B then

```

```

I := 42
else
  I := 0;
Writeln(I);      (*I muss im else-Abschnitt zugewiesen werden
end;

procedure CaseStatement;
var
  I: Integer;
begin
  case C of
    Red..Blue: I := 42;
    else        I := 0;
  end;
  Writeln(I);      (*I muss im else-Abschnitt zugewiesen werden*)
end;

procedure TryStatement;
var
  I: Integer;
begin
  I := 0;
  try
    I := 42;
  except
    Writeln('Should not get here!');
  end;
  Writeln(I);      (*I muss vor try-Abschnitt zugewiesen werden*)
end;

begin
  B := False;
end.

```

Die Lösung besteht entweder darin, Zuweisungen zu allen Quelltextpfaden hinzuzufügen, oder die Variable bereits vor einer Bedingungsanweisung oder einem try-except-Konstrukt zuzuweisen.

4.1.3.6.496 Auf Element 0 kann nicht zugegriffen werden - 'Length' oder 'SetLength' verwenden (E2157)

Der Delphi-Typ string speichert die Länge des Strings nicht in Element 0. Die alte Methode, die Länge eines Strings durch Zugriff auf Element 0 zu verändern oder abzurufen, funktioniert bei langen Strings nicht.

```

program Produce;
var
  str : String;
  len : Integer;
begin
  str := 'Kojo no tsuki';
  len := str[0];
end.

```

Hier versucht das Programm, die Länge des Strings durch direkten Zugriff auf das erste Element abzurufen. Dies ist nicht zulässig.

```

program Solve;
var
  str : String;
  len : Integer;
begin

```

```

str := 'Kojo no tsuki';
len := Length(str);
end.

```

Mit den Standard-Prozeduren SetLength und Length besitzen Sie denselben Funktionsumfang wie durch direkten Zugriff auf das erste Element des Strings. Wenn die Hinweisfunktion aktiviert ist, erhalten Sie eine Warnmeldung darüber, dass der Wert von "len" nicht benutzt wird.

4.1.3.6.497 New wird für dynamische Arrays nicht unterstützt - es muss Setlength verwendet werden (E2255)

Das Programm hat versucht, die Standardprozedur New für ein dynamisches Array zu verwenden. Zum Zuweisen dynamischer Arrays ist korrekterweise die Standardprozedur SetLength zu verwenden.

```

program Produce;
var
  arr : array of integer;

begin
  new(arr, 10);
end.

```

Die Standardprozedur New kann für dynamische Arrays nicht verwendet werden.

```

program Solve;
var
  arr : array of integer;

begin
  SetLength(arr, 10);
end.

```

Verwenden Sie stattdessen zur Zuweisung dynamischer Arrays die Standardprozedur SetLength.

4.1.3.6.498 Nicht verfügbarer Wert (E2142)

Sie haben versucht, einen Wert anzeigen zu lassen, auf den aus dem integrierten Debugger nicht zugegriffen werden kann. Bestimmte Arten von Werten (wie ein String vom Typ Variant mit Nullänge) können nicht aus dem Debugger heraus angezeigt werden.

4.1.3.6.499 An '%s' zugewiesener Wert wird niemals benutzt (H2077)

Diese Hinweismeldung wird angezeigt, wenn der einer Variablen zugewiesene Wert nicht verwendet wird. Bei aktiverter Optimierung wird die Anweisung automatisch entfernt.

Das Problem kann auftreten, wenn die Variable nicht mehr verwendet oder vor ihrer Verwendung erneut zugewiesen wird.

```

program Produce;
(*$HINTS ON*)

procedure Simple;
var
  I: Integer;
begin
  I := 42;           (*--- Hier der Hinweis*)
end;

procedure Propagate;
var
  I: Integer;

```

```

K: Integer;
begin
  I := 0;           (*--- Hier der Hinweis*)
  Inc(I);          (*--- Hier der Hinweis*)
  K := 42;
  while K > 0 do begin
    if Odd(K) then
      Inc(I);          (*--- Hier der Hinweis*)
    Dec(K);
  end;
end;

procedure TryFinally;
var
  I: Integer;
begin
  I := 0;           (*--- Hier der Hinweis*)
  try
    I := 42;
  finally
    Writeln('Reached finally');
  end;
  Writeln(I);        (*Gibt immer 42 aus - bei einer Exception,
                      wird die Anweisung nicht ausgeführt*)
end;

begin
end.

```

In der Prozedur Propagate erkennt der Compiler, dass die Variable I nach der while-Schleife nicht verwendet wird. Sie muss also in der Schleife nicht inkrementiert werden, und somit sind auch die Erhöhung und die Zuweisung vor der Schleife überflüssig.

In TryFinally wird die Zuweisung zu I vor dem try-finally-Block nicht benötigt. Da bei einer Exception die Anweisung Writeln am Ende nicht ausgeführt wird, ist der Wert von I nicht von Bedeutung. Wenn keine Exception auftritt, hat I in der Writeln-Anweisung immer den Wert 42. Somit wird durch die erste Zuweisung die Funktionsweise der Prozedur nicht geändert, und sie kann problemlos entfernt werden.

Diese Hinweismeldung bedeutet nicht, das Ihr Programm einen Fehler enthält. Der Compiler hat lediglich eine Zuweisung entdeckt, die nicht notwendig ist.

Sie können die Zuweisung einfach löschen. Sie wird bei aktiver Optimierung sowieso aus dem kompilierten Code entfernt.

Manchmal besteht das wirkliche Problem aber darin, dass Sie die falsche Variable zugewiesen haben (z. B. J statt I). Prüfen Sie daher die betreffende Zuweisung vor dem Entfernen.

4.1.3.6.500 Nur externe cdecl-Funktionen dürfen varargs verwenden (E2277)

Sie haben versucht, eine varargs-Routine zu implementieren. Dies ist nicht möglich, Sie können nur externe varargs aufrufen.

4.1.3.6.501 Variable erwartet (E2088)

Diese Fehlermeldung wird angezeigt, wenn Sie eine absolute Variable deklarieren, der Direktive absolute jedoch weder eine Integer-Konstante noch ein Variablenname folgt.

```

program Produce;

var
  I: Integer;
  J : Integer absolute Addr(I);  (*--- Hier die Fehlermeldung*)

begin

```

```

end.
program Solve;

const
  Addr = 0;

var
  I: Integer;
  J : Integer absolute I;

begin
end.

```

4.1.3.6.502 Auf Variable '%s' kann wegen Optimierung nicht zugriffen werden (E2171)

Die Auswertungs- oder Überwachungsanweisung versucht, den Wert von <Element> abzurufen, aber der Compiler konnte feststellen, dass die tatsächliche Lebensdauer der Variablen vor diesem Prüfungszeitpunkt beendet war. Dieser Fehler tritt häufig auf, wenn der Compiler feststellt, dass einer lokalen Variable ein Wert zugewiesen wird, der nach einem bestimmten Punkt im Steuerfluss des Programms nicht mehr benutzt wird.

Erstellen Sie eine neue Anwendung.

Platzieren Sie eine Schaltfläche im Formular.

Klicken Sie doppelt auf die Schaltfläche aus, um zur Klick-Methode zu gelangen.

Fügen Sie eine globale Variable 'c' vom Typ Integer zum Implementierungsabschnitt hinzu.

Die Klick-Methode sollte folgendermaßen aussehen:

```

procedure TForm1.Button1Click(Sender: TObject);
  var a, b : integer;
begin
  a := 10;
  b := 20;
  c := b;
  a := c;
end;

```

Setzen Sie einen Haltepunkt bei der zweiten Zuweisung zu 'c'.

Compiler Sie die Anwendung, und führen Sie sie aus.

Klicken Sie auf die Schaltfläche.

Wenn der Haltepunkt erreicht wird, öffnen Sie den Auswerter (Start/Auswerten/Ändern).

Werten Sie A aus.

Der Compiler bemerkt, dass die erste Zuweisung zu 'a' "tot" ist, da der Wert niemals benutzt wird. Aus diesem Grunde verzögert der Compiler sogar die Benutzung von 'a', bis die zweite Zuweisung erfolgt – bis zu dem Zeitpunkt, an dem 'c' zu 'a' zugewiesen wird, wird die Variable 'a' als "tot" erachtet und kann vom Auswerter nicht benutzt werden.

Die einzige Lösung liegt darin, nur Variablen anzeigen zu lassen, für die feststeht, dass sie "lebendige" Werte enthalten.

4.1.3.6.503 Die Typen der tatsächlichen und formalen Var-Parameter müssen übereinstimmen (E2033)

Bei einem Variablenparameter muss das eigentliche Argument genau denselben Typ wie der formale Parameter aufweisen.

```

program Produce;

procedure SwapBytes(var B1, B2: Byte);
var
  Temp: Byte;
begin
  Temp := B1; B1 := B2; B2 := Temp;

```

```

end;

var
  C1, C2: 0..255;      (*Dem Typ Byte ähnlich, jedoch NICHT identisch*)
begin
  SwapBytes(C1,C2);   (*--- Hier die Fehlermeldung*)
end.

```

Die Argumente C1 und C2 sind für SwapBytes nicht annehmbar, obwohl sie genau dieselbe Speicherdarstellung und denselben Bereich wie ein Byte aufweisen.

```

program Solve;

procedure SwapBytes(var B1, B2: Byte);
var
  Temp: Byte;
begin
  Temp := B1; B1 := B2; B2 := Temp;
end;

var
  C1, C2: Byte;
begin
  SwapBytes(C1,C2);   (*--- Hier keine Fehlermeldung*)
end.

```

Sie müssen also tatsächlich C1 und C2 als Bytes deklarieren, damit dieses Beispiel compiliert werden kann.

4.1.3.6.504 Unit %s wurde mit einer unterschiedlichen Version von %s.%s compiliert (F2051)

Dieser schwerwiegende Fehler tritt auf, wenn die Deklaration eines im interface-Abschnitt einer Unit definierten Symbols geändert wurde und der Compiler eine Unit nicht neu compilieren kann, die diese Deklaration benötigt, weil der Quelltext nicht verfügbar ist.

Es gibt mehrere Lösungsmöglichkeiten. Compilieren Sie Unit1 erneut (vorausgesetzt, der Quelltext ist verfügbar), verwenden Sie eine ältere Version von Unit2 bzw. ändern Sie Unit2, oder besorgen Sie sich eine neue Version von Unit1 von der Person, die den Quelltext hat.

Der Fehler kann auch auftreten, wenn eine Unit in Ihrem Projekt den gleichen Namen wie eine Delphi-Standard-Unit hat.

Dies kann beispielsweise passieren, wenn Sie ein Projekt compilieren, das mit einer früheren Delphi-Version erstellt wurde, in der diese Unit nicht vorhanden war (die Unit search.pas war beispielsweise nicht Teil von Delphi 3).

So lösen Sie dieses Problem:

1. Öffnen Sie <Unit2>, und speichern Sie die Datei unter einem neuen Namen.
2. Ändern Sie alle Verweise auf <Unit2> in den uses-Klauseln.
3. Löschen Sie die alten Versionen der Unit (<Unit2>.pas UND <Unit2>.dcu).
4. Compilieren Sie das Projekt erneut.

4.1.3.6.505 Virtuelle Methoden sind in Record-Typen nicht zulässig (E2379)

Für diese Fehler- oder Warnmeldung stehen keine weiteren Informationen zur Verfügung.

4.1.3.6.506 Void-Typ in diesem Kontext nicht verwendbar (E2423)

Der Systemtyp Void darf in einigen Kontexten nicht verwendet werden. Der folgende Beispielcode demonstriert die Kontexte, in

welchen der Typ Void nicht verwendet werden darf.

```
program Project3;

{$APPTYPE CONSOLE}

type
  TBar = class
    property Bar: Void;

  end;

  TBaz = type Void;

var
  Tfoo: ^Void;

procedure Bar(Arg: Void);
begin
end;

function Foo: Void;
begin
end;

end.
```

4.1.3.6.507 \$WEAKPACKAGEUNIT '%s' darf keinen Initialisierungs- oder Finalisierungscode enthalten (E2221)

Eine mit der Direktive \$WEAKPACKAGEUNIT markierte Unit darf weder Quelltext für die Initialisierung bzw. Instanzauflösung noch globale Daten enthalten, weil in einer Anwendung mehrere Kopien dieser Unit vorhanden sein können. Wenn auf die Daten dieser Unit Bezug genommen wird, führt dies zu einem mehrdeutigen Ergebnis. Diese Mehrdeutigkeit vergrößert sich noch, wenn dynamisch geladene Packages in der Anwendung eingesetzt werden.

```
(*$WEAKPACKAGEUNIT*)
unit yamadama;
interface
implementation
  var
    Title : String;

  initialization
    Title := 'Tiny Calc';
  finalization
end.
```

In diesem Beispiel treten zwei Probleme auf: Title ist eine globale Variable und wird außerdem im initialization-Abschnitt der Unit initialisiert.

Dieses Problem kann auf zwei Arten gelöst werden: Entfernen Sie entweder die Direktive \$weakpackageunit aus der Unit, oder löschen Sie alle globalen Daten sowie den Programmtext für die Initialisierung und die Finalisierung.

4.1.3.6.508 \$WEAKPACKAGEUNIT '%s' enthält globale Daten (E2203)

Eine mit \$WEAKPACKAGEUNIT markierte Unit wird in ein Package eingefügt, enthält jedoch globale Daten. Eine derartige Unit darf aber weder globale Daten noch Quelltext zur Initialisierung oder Finalisierung enthalten.

Die einzige Lösung für dieses Problem besteht darin, die Direktive \$WEAKPACKAGEUNIT oder die globalen Daten von der Unit zu entfernen, bevor diese in das Package übernommen wird.

4.1.3.6.509 In Set-Ausdruck WideChar auf Byte-Char verringert (W1050)

"Set of char" definiert in Win32 ein Set über den gesamten Bereich des Char-Typs. Da Char in Win32 ein byte-großer Typ ist, wird dadurch ein Set von maximaler Größe mit 256 Elementen definiert. In .NET ist Char ein wortgroßer Typ, dessen Bereich (0..65535) die Kapazität des Set-Typs überschreitet.

Bei vorhandenem Code, der die Syntax "Set of Char" verwendet, behandelt der Compiler den Ausdruck als "set of AnsiChar". Die Warnmeldung erinnert Sie daran, dass das Set nur den Booleschen Status von 256 verschiedenen Elementen, aber nicht den vollen Bereich des Char-Typs speichern kann.

4.1.3.6.510 Falsche oder beschädigte RLINK32.DLL (E2152)

Die interne Konsistenzprüfung für die Datei RLINK32.DLL ist fehlgeschlagen.

Wenden Sie sich an CodeGear, wenn Sie diese Fehlermeldung erhalten.

4.1.3.6.511 Operator ist auf diesen Operandentyp nicht anwendbar (E2015)

Diese Fehlermeldung wird ausgegeben, wenn ein Operator nicht auf die Operanden angewandt werden kann, denen er übergeben wurde – beispielsweise wenn ein Boolescher Operator auf einen Zeiger angewandt wird.

```
program Produce;
var
  P: ^Integer;
begin
  if P and P^ > 0 then
    Writeln('P points to a number greater 0');
end.
```

Hier war sich ein C++ Programmierer über die Operatorpriorität in Delphi nicht im klaren; P ist kein Boolescher Ausdruck, und der Vergleich muss in runde Klammern gesetzt werden.

```
program Solve;
var
  P: ^Integer;
begin
  if (P <> nil) and (P^ > 0) then
    Writeln('P points to a number greater 0');
end.
```

Wenn wir P explizit mit nil vergleichen und runde Klammern verwenden, ist der Compiler rundum zufrieden.

4.1.3.6.512 XML-Kommentar bei '%s' hat das cref-Attribut '%s', das nicht aufgelöst werden konnte (W1206)

Diese Warnung wird ausgegeben, wenn das Attribut cref im XML nicht aufgelöst werden kann.

Dies ist eine Warnung bei der Verarbeitung der XML-Dokumentation. Das XML ist wohlgeformt, aber die Bedeutung des Kommentars ist unsicher. XML cref-Referenzen folgen dem .NET-Stil. Details finden Sie unter <http://msdn2.microsoft.com/en-us/library/acd0tfbe.aspx>. Der Build-Prozess wird durch eine Dokumentationswarnung nicht verhindert.

4.1.3.6.513 XML-Kommentar in '%s' ist falsch strukturiert -- 'Das Zeichen '%c' wurde erwartet.' (W1205)

Diese Warnung wird ausgegeben, wenn das erwartete Zeichen im XML nicht gefunden wird.

Dies ist ein Fehler bei der XML-Dokumentationsverarbeitung. Das XML ist nicht wohlgeformt. Dies ist eine Warnung, weil der Build-Prozess durch einen Dokumentationsfehler nicht verhindert wird.

4.1.3.6.514 XML-Kommentar in '%s' ist falsch strukturiert -- 'Ein Name enthält ein ungültiges Zeichen.' (W1204)

Diese Warnung wird ausgegeben, wenn ein Name im XML ein ungültiges Zeichen enthält.

Dies ist ein Fehler bei der XML-Dokumentationsverarbeitung. Das XML ist nicht wohlgeformt. Dies ist eine Warnung, weil der Build-Prozess durch einen Dokumentationsfehler nicht verhindert wird.

4.1.3.6.515 XML-Kommentar in '%s' ist falsch strukturiert -- 'Ein Name beginnt mit einem ungültigen Zeichen.' (W1203)

Diese Warnung wird ausgegeben, wenn ein XML-Name mit einem ungültigen Zeichen beginnt.

Dies ist ein Fehler bei der XML-Dokumentationsverarbeitung. Das XML ist nicht wohlgeformt. Dies ist eine Warnung, weil der Build-Prozess durch einen Dokumentationsfehler nicht verhindert wird.

4.1.3.6.516 Parameter '%s' hat kein zugehöriges param-Tag im XML-Kommentar für '%s' (aber andere Parameter haben dieses Tag) (W1208)

Diese Warnung wird ausgegeben, wenn ein XML-Parameter im XML-Kommentar im Gegensatz zu anderen Parametern über kein zugehöriges param-Tag verfügt.

Dies ist eine Warnung bei der Verarbeitung der XML-Dokumentation. Es ist mindestens ein Tag vorhanden, aber einige Parameter in der Methode haben kein Tag. Der Build-Prozess wird durch eine Dokumentationswarnung nicht verhindert.

4.1.3.6.517 XML-Kommentar bei '%s' hat ein param-Tag für '%s', aber es gibt keinen Parameter mit diesem Namen (W1207)

Diese Warnung wird ausgegeben, wenn das XML ein Parameter-Tag für einen nicht vorhandenen Parameter enthält.

Dies ist eine Warnung bei der Verarbeitung der XML-Dokumentation. Das XML ist wohlgeformt, aber es wurde ein Tag für einen Parameter erstellt, der in einer Methode nicht vorhanden ist. Der Build-Prozess wird durch eine Dokumentationswarnung nicht verhindert.

4.1.3.6.518 XML-Kommentar in '%s' ist falsch strukturiert -- 'Referenz auf nicht definierte Entität '%s'.' (W1202)

Diese Warnung wird ausgegeben, wenn das XML eine nicht definierte Entität referenziert.

Dies ist ein Fehler bei der XML-Dokumentationsverarbeitung. Das XML ist nicht wohlgeformt. Dies ist eine Warnung, weil der Build-Prozess durch einen Dokumentationsfehler nicht verhindert wird.

4.1.3.6.519 XML-Kommentar in '%s' ist falsch strukturiert -- 'Whitespace ist an dieser Position nicht zulässig.' (W1201)

Diese Warnung wird ausgegeben, wenn der Compiler einen Whitespace an einer Position entdeckt, an der Whitespace nicht zulässig sind.

Dies ist ein Fehler bei der XML-Dokumentationsverarbeitung. Das XML ist nicht wohlgeformt. Dies ist eine Warnung, weil der Build-Prozess durch einen Dokumentationsfehler nicht verhindert wird.

4.1.3.6.520 Konstante 0 wurde zu NIL konvertiert (W1013)

Der Delphi-Compiler erlaubt es nun, die Konstante 0 in Zeigerausdrücken anstelle von nil zu verwenden. Dadurch kann alter Quelltext compiliert werden, der Änderungen in der RTL enthält.

```
program Produce;

procedure p0(p : Pointer);
begin
end;

begin
  p0(0);
end.
```

In diesem Beispiel wurde die Prozedur p0 so deklariert, dass sie als Parameter Zeiger erwartet. Allerdings wird die Konstante 0 übergeben. Der Compiler führt die erforderlichen Konvertierungen intern durch und ändert dabei 0 zu nil. Dadurch wird sichergestellt, dass der Quelltext korrekt ausgeführt werden kann.

```
program Solve;

procedure p0(p : Pointer);
begin
end;

begin
  p0(NIL);
end.
```

Dieses Problem kann auf zwei Arten gelöst werden. Im obigen Beispiel wurde die Konstante 0 durch nil ersetzt. Alternativ dazu könnte auch die Prozedurdefinition geändert werden, sodass der Parametertyp vom Typ Integer ist.

4.1.3.6.521 Disposed_ darf nicht in Klassen mit Destruktoren deklariert werden

Für diese Meldung oder Warnung steht keine weiteren Hilfe zur Verfügung.

4.2 Together-Referenz

Dieser Abschnitt enthält Links zum Referenzmaterial für die UML-Modellierung mit Together.

4.2.1 Together-Glossar

Dieses Thema enthält eine Erklärung wichtiger Begriffe, die in der Benutzeroberfläche und der Dokumentation von Together verwendet werden. Das Glossar ist alphabetisch sortiert.

Begriff	Beschreibung
Kardinalität	Die Anzahl der Elemente in einer Gruppe. Siehe auch <i>Multiplizität</i> .
Klassifizierer	Ein Klassifizierer ermöglicht eine Zuordnung von Instanzen. Er beschreibt eine Gruppe von Instanzen, die über gemeinsame Merkmale verfügen. In Together sind Klassifizierer die Hauptknoten von Klassendiagrammen: Klasse, Interface, Struktur, Delegat, Enum und Modul. Einige dieser Knoten können über innere Klassifizierer verfügen, d.h. weitere Klassifizierer enthalten (siehe Innere Klassifizierer (siehe Seite 1298)).
Bereich	Bestimmte Together-Modellelemente (i.d.R. Klassen) werden durch Rechtecke dargestellt, die mehrere Bereiche enthalten. Die Darstellung der Bereiche kann geändert werden. Weitere Informationen finden Sie unter Diagrammoptionen (Erscheinungsbild) (siehe Seite 1282).
Diagramm	Eine grafische Darstellung einer Gruppe von Modellelementen. In der Regel werden zur Darstellung verbundene Bögen (Beziehungen) und Eckpunkte (andere Modellelemente) verwendet. Welche Diagramme für ein Projekt verfügbar sind, hängt vom Projekttyp ab.
Aufrufsspezifikation	Siehe Aufrufsspezifikation (siehe Seite 1320).
Modellelement	Unter einen Modellelement versteht man eine beliebige Komponente des Modells, die in ein Diagramm eingefügt werden kann. Modellelemente werden als Knoten und Beziehungen dargestellt
Multiplizität	Eine Spezifikation für den Bereich der Kardinalitäten, die eine Gruppe annehmen kann. Multiplizitätsspezifikationen können für Assoziationsenden, Teile innerhalb von Zusammensetzungen, Wiederholungen und für andere Zwecke definiert werden. Eine Multiplizität ist eine Teilmenge der nicht negativen ganzen Zahlen. Siehe auch <i>Kardinalität</i> .
N-fache Assoziation	Eine Assoziation zwischen drei oder mehr Klassen. Jede Instanz der Assoziation ist n-Tupel der Werte aus den entsprechenden Klassen.
Verknüpfung	Eine Verknüpfung repräsentiert ein Knotenelement, das im selben oder in einem anderen Diagramm vorhanden ist.
Ansichtsfilter	Ein Mechanismus zum Ein- und Ausblenden einer bestimmten Gruppe von Modellelementen. Wenn Sie an großen Projekten arbeiten, können Ihre Diagramme sehr unübersichtlich werden. Sie können in Together Informationen selektiv ein- und ausblenden. Siehe Ansichtsfilter verwenden (siehe Seite 194).

Siehe auch[Hilfe zur Hilfe \(siehe Seite 1378\)](#)[Allgemeines zu Together \(siehe Seite 1380\)](#)

4.2.2 GUI-Komponenten für die Modellierung

In diesem Abschnitt werden die Komponenten beschrieben, die in der RAD Studio-Oberfläche für die UML-Modellierung zur Verfügung stehen.

4.2.2.1 Menüs

Menü	Beschreibung	
Datei	Sie können über das Menü Datei Diagramme in Bilddateien exportieren und drucken.	
Bearbeiten	Mit den Befehlen im Menü Bearbeiten können Sie Diagramme und Diagrammelemente ausschneiden, kopieren und einfügen, alle Elemente in einem Diagramm auswählen sowie Aktionen rückgängig machen oder wiederholen.	
Ansicht	Das Menü Ansicht enthält den Befehl zum Öffnen der Modellansicht.	
Projekt	Im Menü Projekt können Sie die Together-Unterstützung für bestimmte Projekte in der aktuell geöffneten Projektgruppe aktivieren und deaktivieren.	
Refactor	Das Menü Refactor enthält die Refactoring-Befehle für die Implementierungsprojekte.	
Tools	Mit den Befehlen im Menü Tools können Sie Dokumentation erzeugen, die Pattern-Registrierung und den Pattern-Organizer öffnen, Audits und Metriken ausführen (für Implementierungsprojekte) sowie die Together-spezifischen Optionen festlegen.	
Kontextmenü Diagramms	des	Über das Kontextmenü des Diagramms können Sie neue Elemente hinzufügen, das Layout verwalten, die Ansicht vergrößern oder verkleinern, Diagrammelemente ein- oder ausblenden, das Diagramm mit der Modellansicht synchronisieren und Hyperlinks bearbeiten.
Kontextmenü Modellansicht	der	Die Kontextmenüs der verschiedenen Elemente in der Modellansicht sind kontextbezogen. Welche Elemente Sie einem Diagramm aus der Modellansicht hinzufügen können, richtet sich nach dem jeweiligen Element- und Diagrammtyp.
Kontextmenüs Elemente	für	Über die Kontextmenüs der einzelnen Diagrammelemente können Sie Member hinzufügen und löschen (oder das Element selbst entfernen), Elemente ausschneiden, kopieren und einfügen, Quelltext und Elementinformationen anzeigen usw. Klicken Sie mit der rechten Maustaste auf verschiedene Elemente, um festzustellen, welche Befehle das jeweilige Kontextmenü enthält.

Siehe auch[Modellansicht \(siehe Seite 1270\)](#)[Diagrammansicht \(siehe Seite 1269\)](#)

4.2.2.2 Tool-Palette

[Ansicht>Tool-Palette](#)

Together erweitert die Tool-Palette von RAD Studio um Modellelemente.

Die Tool-Palette von RAD Studio enthält spezielle Registerkarten für die unterstützten UML-Diagramme. Wenn Sie in der Diagrammansicht ein Diagramm öffnen, wird die zugehörige Registerkarte in der Tool-Palette angezeigt.

Die Tool-Palette enthält Schaltflächen für die Modellelemente (Knoten, Beziehungen), die in das aktuelle Diagramm eingefügt werden können. Sie können aber auch festlegen, dass die Registerkarten für alle Diagrammtypen angezeigt werden. Mit den Schaltflächen in der Tool-Palette können Sie Ihren Diagrammen die gewünschten Elemente hinzufügen.

Anmerkung: Welche Modellelemente zur Verfügung stehen, richtet sich nach dem Typ des Diagramms, das in der Diagrammansicht ausgewählt ist. Eine Beschreibung der verfügbaren Elemente finden Sie in der Together-Referenz.

Tip: Sie können festlegen, welche Diagrammelemente in der Tool-Palette angezeigt werden und eigene Tool-Paletten-Registerkarten mit bestimmten Schaltflächen erstellen. Die Einträge können zur individuellen Anpassung der Tool-Palette ausgeschnitten, kopiert, eingefügt, gelöscht, umbenannt und nach oben oder unten verschoben werden. Sie können auch alphabetisch sortiert und als Liste angezeigt werden. Diese Operationen werden über das Kontextmenü der Tool-Palette ausgeführt. Weitere Informationen finden Sie in der RAD Studio-Dokumentation.

Siehe auch

Ein Element erstellen (siehe Seite 142)

Eine einfache Beziehung erstellen (siehe Seite 141)

4.2.2.3 Objektinspektor

Ansicht > Objektinspektor

Wenn die Together-Unterstützung aktiviert ist, werden im Objektinspektor die Eigenschaften des Elements angezeigt, das Sie in der Modell- oder Diagrammansicht ausgewählt haben. Um den Objektinspektor zu öffnen, wählen Sie im Menü Ansicht den Befehl Objektinspektor. Alternativ können Sie F4 oder ALT+EINGABE drücken. Der Inhalt des Objektinspektors ist vom ausgewählten Elementtyp abhängig.

Sie können im Objektinspektor die Diagramm- oder Elementeigenschaften überprüfen und ändern.

Es gibt mehrere Kategorien von Eigenschaften:

Element	Beschreibung
Beschreibung	Ein Textfeld, in das Sie optional eine Beschreibung des Elements eingeben können.
Design	Mit diesen Eigenschaften können Sie festlegen, wie das Element im Diagramm angezeigt wird.
Allgemein	Allgemeine Eigenschaften für UML- und Quelltextelemente.
Benutzereigenschaften	Dieser Knoten wird angezeigt, wenn Sie Benutzereigenschaften definiert haben. Sie können beliebig Benutzereigenschaften definieren.

Sie können im Objektinspektor die Eigenschaften eines Elements überprüfen und bearbeiten. Wenn Sie auf ein Feld klicken, sehen Sie, welcher interne Editortyp zur Verfügung steht: Textbereich, Kombinationsfeld mit Werteliste oder Dialogfeld. Schreibgeschützte Felder werden grau angezeigt. Wenn Sie auf eine Eigenschaft klicken, wird die Beschreibung unten im Objektinspektor angezeigt.

Siehe auch

Benutzereigenschaften verwenden (siehe Seite 135)

Optionswert-Editoren (siehe Seite 1280)

4.2.2.4 Diagrammansicht

Kontextmenü (in der Modellansicht) ▶ Diagramm öffnen

In der Diagrammansicht werden Modelldiagramme angezeigt. Jedes Diagramm befindet sich auf einer eigenen Registerkarte.

Zum Öffnen der Diagrammansicht wählen Sie in der Modellansicht ein Diagramm, einen Namespace oder ein Package aus, klicken mit der rechten Maustaste und wählen im Kontextmenü den Befehl Diagramm öffnen.

Die meisten Operationen mit Diagrammelementen und Beziehungen werden durch Ziehen mit der Maus oder über die Befehle in den Kontextmenüs ausgeführt.

Über die Kontextmenüs können beispielsweise folgende Aktionen durchgeführt werden:

- Hinzufügen oder Löschen von Diagrammelementen und Beziehungen
- Hinzufügen oder Löschen von Membern in Elementen
- Erstellen von Elementen nach Pattern
- Ausschneiden, Kopieren und Einfügen ausgewählter Elemente
- Wechseln zum Quelltext
- Verknüpfen von Diagrammen durch Hyperlinks
- Vergrößern und Verkleinern der Ansicht

Element	Beschreibung
Arbeitsbereich	Den größten Teil der Diagrammansicht nimmt das aktuelle Diagramm ein.
Kontextmenü	Die Kontextmenüs der Diagrammansicht sind kontextabhängig. Wenn Sie mit der rechten Maustaste auf ein Modellelement klicken, wird das Kontextmenü mit den speziellen Befehlen für dieses Element angezeigt (auch Klassen-Member können auf diese Weise bearbeitet werden). Wenn Sie mit der rechten Maustaste auf den Diagrammhintergrund klicken, wird das Kontextmenü für das Diagramm geöffnet.
Übersicht (Schaltfläche)	Der Übersichtsbereich wird geöffnet (siehe unten).

Übersichtsbereich

Mit der Übersichtsfunktion der Diagrammansicht können Sie eine Miniaturansicht des aktuellen Diagramms anzeigen. Die Schaltfläche **Übersicht** befindet sich in der rechten oberen Ecke jedes Diagramms.

OCL-Editor

Mit dem OCL-Editor können OCL-Ausdrücke eingegeben und bearbeitet werden. Die Namen der Modellelemente (Klassen, Operationen, Attribute usw.), die Sie in den Ausdrücken verwenden, werden bei einer Änderung automatisch aktualisiert. Dadurch ist gewährleistet, dass die OCL-Einschränkungen immer auf dem neuesten Stand sind.

Siehe auch

Diagramme – Überblick (siehe Seite 1447)

Überblick zur OCL-Unterstützung (siehe Seite 1453)

Diagramme erstellen (siehe Seite 116)

Modellansicht (siehe Seite 262)

4.2.2.5 Modellansicht

Ansicht ▶ Modellansicht

Verwenden Sie das Fenster der Modellansicht zum Anzeigen der logischen Struktur und der Hierarchie Ihres Projektes. Beachten Sie bitte, dass das ECO-Framework im Architect SKU und höher verfügbar ist; ECO-bezogene Symbole und Themenverknüpfungen stehen in anderen Produkt-SKUs nicht zur Verfügung.

Code-Visualisierungssymbol	Bedeutung
	Ein Projekt
📁	Ein UML-Package (ECO-Framework)
📄	Ein UML-Package-Symbol (Code-Visualisierung)
🔴	Eine Klasse (ECO-Framework)
🟡	Eine Klasse (Code-Visualisierung)
🔵	Ein Interface (Code-Visualisierung)
🟠	Eine Operation (ECO-Framework)
🟠	Eine Operation (Code-Visualisierung)
🟠	Eine Eigenschaft (ECO-Framework)
🟠	Eine Eigenschaft einer Klasse (Code-Visualisierung)
	Das Diagramm für das Projekt oder UML-Package
↗	Eine Beziehung zu einer anderen Klasse oder einem Interface (Code-Visualisierung)
⤒	Generalisierung (ECO-Framework)
⤒	Assoziation (ECO-Framework)
⤒	Abgeleitete Assoziation (ECO-Framework)

Anmerkung: In Code-Visualisierungsdiagrammen entspricht jede .NET-Namespace-Deklaration einem UML-Package (dies gilt *nicht* für ECO-fähigen Quelltext). Durch Doppelklicken auf einen Namespace-Knoten in der Modellhierarchie wird keine spezielle Quelltextdatei geöffnet, weil Namespaces sich auf mehrere Quelldateien erstrecken können.

Tip: Um für eine bestimmte Klasse, ein Interface oder ein Member schnell den Quelltext-Editor zu öffnen, doppelklicken Sie in der Modellhierarchie auf das Element.

Siehe auch

[Verwenden der Code-Visualisierung](#)

[Verwenden des Modellansichtfensters und des Code-Visualisierungsdiagramms](#)

[Verwenden des Übersichtsfensters](#)

[UML-Features in Delphi für .NET](#)

[Überblick zum ECO-Framework](#)

[Integrierte Modellierungstools - Übersicht](#)

[Verwenden der ECO-Experten](#)

[Importieren eines Modells](#)

[Den ECO-Space-Designer verwenden](#)

OCL-Ausdruckseditor verwenden

Erstellen einer ECO-fähigen Benutzeroberfläche

Deployment von ECO-Anwendungen

4.2.2.6 GUI-Komponenten für Pattern

In diesem Abschnitt werden die Komponenten beschrieben, mit denen in der RAD Studio-Oberfläche auf die Pattern-Funktionen von Together zugegriffen werden kann.

4.2.2.6.1 Pattern-Organizer

Tools > Pattern-Organizer

Im Pattern-Organizer können die im Pattern-Experten vorhandenen Pattern mit Hilfe virtueller Hierarchien, Ordner und Verknüpfungen logisch organisiert werden. Außerdem können Sie im Pattern-Organizer Eigenschaften von Pattern anzeigen und bearbeiten.

Virtuelle Pattern-Hierarchien

In diesem Bereich wird die logische Hierarchie der Pattern angezeigt. Für den Stammordnerknoten, die Unterordner und die Pattern-Elemente stehen Kontextmenüs zur Verfügung. Das Kontextmenü für den Stammordner enthält folgende Befehle:

Menüelement	Beschreibung
Neue Pattern-Hierarchie	Mit diesem Befehl wird ein neuer Pattern-Hierarchieknoten erstellt.
Ordner sortieren	Die Knoten werden in aufsteigender alphabetischer Reihenfolge sortiert.

Das Kontextmenü für die Unterordner unterhalb des Stammordners enthält folgende Befehle:

Menüelement	Beschreibung
Neuer Ordner	Mit diesem Befehl wird unterhalb des markierten Ordners ein neuer Unterordner erstellt.
Neue Verknüpfung	Die Pattern-Registrierung wird geöffnet, in der Sie eine neue Verknüpfung für ein Pattern erstellen können. Die Verknüpfung wird in den ausgewählten Ordner eingefügt.
Ausschneiden	Der ausgewählte Knoten wird entfernt und in die Zwischenablage kopiert.
Kopieren	Der ausgewählte Knoten wird in die Zwischenablage kopiert.
Einfügen	Der Inhalt der Zwischenablage wird in den ausgewählten Knoten eingefügt.
Löschen	Der ausgewählte Knoten wird gelöscht.
Ordner sortieren	Die Knoten werden in aufsteigender alphabetischer Reihenfolge sortiert.

Das Kontextmenü für Pattern-Elemente enthält die folgenden Befehle:

Pattern zuweisen	Die Pattern-Registrierung wird geöffnet, in der Sie dem ausgewählten Pattern-Element ein Pattern zuweisen können.
Ausschneiden	Der ausgewählte Knoten wird entfernt und in die Zwischenablage kopiert.
Kopieren	Der ausgewählte Knoten wird in die Zwischenablage kopiert.
Einfügen	Der Inhalt der Zwischenablage wird in den ausgewählten Knoten eingefügt.
Löschen	Der ausgewählte Knoten wird gelöscht.

Eigenschaften

In diesem Bereich werden die Eigenschaften des ausgewählten Pattern oder Ordners angezeigt. Die Felder **Name** und **Visible** können geändert werden.

Name	Der Name, der in der virtuellen Pattern-Hierarchie angezeigt wird. Dieses Feld kann bearbeitet werden.
Gültig	Dieses Feld ist nur für Pattern von Bedeutung. Wenn das Pattern mit der Pattern-Registrierung registriert wurde, hat es den Status <i>Gültig</i> . Andernfalls hat das Pattern den Status <i>Ungültig</i> , und es wird nicht im Pattern-Experten angezeigt. Ordner haben immer den Status <i>Gültig</i> und werden im Pattern-Experten angezeigt (sofern sie nicht aufgrund ihrer Eigenschaft <i>Visible</i> verborgen sind).
Visible	Mit diesem Kombinationsfeld legen Sie fest, ob das Pattern oder der Ordner im Pattern-Experten angezeigt wird.

Pattern-Beschreibung

Dieser schreibgeschützte Bereich enthält Kommentare zu dem ausgewählten Pattern.

Verzeichnisse der gemeinsam genutzten Pattern bearbeiten

Klicken Sie auf diese Schaltfläche, um die Liste mit den Verzeichnissen der gemeinsam genutzten Pattern anzuzeigen. Stellen Sie die Liste mit den Schaltflächen Hinzufügen und Entfernen zusammen, und klicken Sie auf OK.

Siehe auch

Überblick über Pattern (↗ siehe Seite 1454)

Pattern-Registrierung (↗ siehe Seite 1272)

Pattern-Experte (↗ siehe Seite 1277)

4.2.2.6.2 Pattern-Registrierung

Pattern-Organizer (Kontextmenü) ▶ Neue Verknüpfung (oder: Pattern zuweisen)

Die Pattern-Registrierung definiert die virtuelle Hierarchie der Pattern. Wenn Sie im Pattern-Organizer einen Ordner oder eine Verknüpfung erstellen und die Änderungen speichern, wird der Registrierungsdatei ein neuer Eintrag hinzugefügt. Alle Operationen mit dem Inhalt der Pattern-Registrierung werden im Pattern-Organizer durchgeführt und mit der Registrierung synchronisiert.

Das Fenster wird geöffnet, wenn Sie im Kontextmenü des Pattern-Organizers den Befehl Neue Verknüpfung oder Pattern zuweisen wählen.

Filter

Mit Hilfe der Filter legen Sie fest, welche Pattern in der Tabelle **Pattern** angezeigt werden. Folgende Filteroptionen stehen im Bereich **Filter** zur Auswahl:

Option	Beschreibung
Kategorie	Folgende Optionen sind verfügbar: Klasse, Beziehung, Member, Andere, Alle
Registriert	Diese Option ermöglicht eine Filterung nach dem Registrierungsstatus: Alle – Alle vorhandenen Pattern. Ja – Pattern, die Verknüpfungen zugeordnet sind. Ohne – Pattern, die keinen Verknüpfungen zugeordnet sind.
Container	Mit dieser Option werden Pattern nach der Container-Metaklasse gefiltert.
Diagrammtyp	Diese Option filtert die Pattern für den gewählten Diagrammtyp heraus.

Sprache	Diese Option steht nur für Implementierungsprojekte zur Verfügung und ermöglicht eine Filterung nach sprachspezifischen Pattern.
---------	--

Hauptbereiche

Option	Beschreibung
Pattern	Eine Tabelle mit den Namen und Typen der verfügbaren Pattern wird angezeigt.
Pattern-Eigenschaften	Die Eigenschaften des Pattern werden angezeigt, das in der Tabelle Pattern ausgewählt ist.
Pattern-Beschreibung	Dieser schreibgeschützte Bereich enthält Kommentare zu dem Pattern, das in der Tabelle Pattern ausgewählt ist.

Schaltflächen

Schaltfläche	Beschreibung
Synchronisieren	Mit dieser Schaltfläche können Sie den gesamten Speicherbereich nach Pattern durchsuchen und die Pattern-Registrierung aktualisieren.
OK	Klicken Sie auf diese Schaltfläche, um die Filtereinstellungen zu speichern und das Fenster zu schließen.
Abbrechen	Wenn Sie auf diese Schaltfläche klicken, werden die Filtereinstellungen verworfen, und das Fenster wird geschlossen.
Hilfe	Diese Seite wird angezeigt.

Siehe auch

Überblick über Pattern ( siehe Seite 1454)

Pattern-Organizer ( siehe Seite 1271)

Pattern-Experte ( siehe Seite 1277)

4.2.2.7 GUI-Komponenten für die Qualitätssicherung

In diesem Abschnitt werden die Komponenten beschrieben, mit denen in der RAD Studio-Oberfläche auf die Qualitätssicherungsfunktionen von Together zugegriffen werden kann.

4.2.2.7.1 Audit-Ergebnis (Fenster)

QA-Audits (Dialogfeld) ▶ Schaltfläche Start

In diesem Fenster können Sie das Audit-Ergebnis anzeigen und exportieren.

Das Audit-Ergebnis wird in der Audits-Ansicht in Tabellenform angezeigt.

Bei jeder Ausführung von Audits für dasselbe Projekt wird das Ergebnis auf einer neuen Registerkarte im Fenster angezeigt.

Das Ergebnisfenster wird zwar frei platzierbar geöffnet, es kann aber angedockt werden. Sie können es an die vier Ränder des RAD Studio-Fensters andocken. Das Fenster kann beliebig positioniert werden.

Tip: Drücken Sie in der Audits-Ansicht die Taste F1, um diese Seite anzuzeigen.

4

Element	Beschreibung
Symbolleistenschalter	
Audit-Ergebnis speichern	Das Audit-Ergebnis wird gespeichert.
Audit-Ergebnis drucken	Das Audit-Ergebnis wird gedruckt.
Aktualisieren	Die angezeigten Einträge werden neu berechnet.
Neu starten	Das Dialogfeld Audits wird geöffnet, in dem Sie neue Einstellungen festlegen und die Audit-Analyse erneut starten können.
Kontextmenübefehle	
Gruppieren nach	Die Tabelleneinträge werden nach der ausgewählten Spalte gruppiert.
Beschreibung anzeigen	Ein Fenster mit dem vollständigen Namen und einer Beschreibung des ausgewählten Audits wird angezeigt.
Öffnen	Das ausgewählte Element wird im Editor geöffnet und der betreffende Quelltext hervorgehoben.
Kopieren	Sie können eine oder mehrere Zeilen im Audit-Ergebnis kopieren. Wenn Sie mehrere Zeilen kopieren möchten, halten Sie beim Klicken die Taste STRG gedrückt.
Schließen	Die aktuelle Registerkarte wird geschlossen.
Alle schließen	Alle Registerkarten und das Ergebnisfenster werden geschlossen.
Alle außer diesem schließen	Alle Registerkarten außer der aktuell angezeigten werden geschlossen.

In der Tabelle werden nur die Verstöße gegen die Audit-Regeln angezeigt. Daher sind nicht notwendigerweise sämtliche ausgeführten Audits und alle verarbeiteten Packages oder Klassen aufgeführt. Die Tabelle enthält folgende Spalten:

Regelverletzungen

Spaltenüberschrift	Beschreibung
Abkürzung	Die Abkürzung des Audit-Namens. Der vollständige Name wird in der Beschreibung angezeigt (wählen Sie dazu im Kontextmenü der Verletzung Beschreibung anzeigen).
Beschreibung	Hier wird beschrieben, warum der Eintrag eine Regelverletzung ist.
Schweregrad	Hier wird angezeigt, wie ernsthaft Verletzungen der Regel sind. Sie können die Tabelle nach dieser Spalte sortieren und abschätzen, welche Verletzungen schwer wiegend sind und welche nicht.
Ressource	Das als Regelverletzung gemeldete Quelltextelement.
Datei	Die Datei mit dem problematischen Quelltext.
Zeile	Die Nummer der Zeile, in der sich der problematische Quelltext befindet.

Siehe auch

Überblick über Qualitätssicherungsfunktionen (siehe Seite 1456)

Audit-Ergebnisse anzeigen (siehe Seite 270)

Ausführen von Audits (siehe Seite 269)

4.2.2.7.2 Metrikergebnis (Fenster)

[QS-Metriken \(Dialogfeld\) ▶ Schaltfläche Start](#)

In diesem Fenster können Sie das Metrikergebnis anzeigen und exportieren.

Metrikergebnisse werden in Tabellenform im Metrikergebnis-Fenster angezeigt. Die Zeilen enthalten die analysierten Elemente, die Spalten die entsprechenden Werte der ausgewählten Metriken. Über die Kontextmenüs der Zeilen und Spalten können Sie zum Quelltext wechseln, Beschreibungen der Metriken anzeigen und eine grafische Ausgabe erstellen.

Tip: Drücken Sie im Metrikergebnis-Fenster die Taste F1, um diese Seite anzuzeigen.

Element	Beschreibung
Symbolleistenschalter	
Speichern	Das Metrikergebnis wird gespeichert.
Aktualisieren	Die angezeigten Einträge werden neu berechnet.
Neu starten	Das Dialogfeld Metriken wird geöffnet, in dem Sie neue Einstellungen festlegen und die Metrikanalyse erneut starten können.
Kontextmenübefehle	
Beschreibung anzeigen	Ein Fenster mit dem vollständigen Namen und einer Beschreibung der ausgewählten Metrik wird angezeigt.
Diagramm	Ein Metrikiagramm wird erstellt.

Siehe auch

Überblick über Qualitätssicherungsfunktionen (siehe Seite 1456)

Metriken ausführen (siehe Seite 273)

Metrikergebnisse anzeigen (siehe Seite 274)

4.2.3 Together-Experten

Dieser Abschnitt beschreibt die Experten, die bei der UML-Modellierung verwendet werden.

4.2.3.1 Neue Projektexperten in Together

Dieser Abschnitt beschreibt die Experten, die verwendet werden, um neue Together-Modellprojekte zu erstellen.

4.2.3.1.1 UML 1.5 Together-Experte für Design-Projekte

Datei>Neu>Weitere>Design-Projekte>UML 1.5-Design-Projekt

Dieses Projekt wird von Together zur Erstellung eines UML 1.5-Design-Projekts bereitgestellt.

Um den Experten zu starten, wählen Sie im Hauptmenü **Datei>Neu>Weitere**. Das Dialogfeld Objektgalerie wird geöffnet. Wählen Sie die Kategorie **Design-Projekte>UML 1.5-Design-Projekt**. Klicken Sie auf OK.

Das Dialogfeld Neue Anwendung wird geöffnet. Geben Sie den Namen und den Pfad des Modellprojekts an. Klicken Sie auf OK.

Das neue UML 1.5-Designprojekt wird erstellt. Verwenden Sie zur Anzeige der Struktur die Modellansicht.

Siehe auch

Erstellen eines Projekts ([siehe Seite 249](#))

Unterstützte Projektformate ([siehe Seite 1332](#))

4.2.3.1.2 UML 2.0 Together-Experte für Design-Projekte

Datei▶**Neu**▶**Weitere**▶**Design-Projekte**▶**UML 2.0-Design-Projekt**

Dieses Projekt wird von Together zur Erstellung eines UML 2.0-Design-Projekts bereitgestellt.

Um den Experten zu starten, wählen Sie im Hauptmenü **Datei**▶**Neu**▶**Weitere**. Das Dialogfeld Objektgalerie wird geöffnet. Wählen Sie die Kategorie **Design-Projekte**▶**UML 2.0-Design-Projekt**. Klicken Sie auf **OK**.

Das Dialogfeld Neue Anwendung wird geöffnet. Geben Sie den Namen und den Pfad des Modellprojekts an. Klicken Sie auf **OK**.

Das neue UML 2.0-Designprojekt wird erstellt. Verwenden Sie zur Anzeige der Struktur die Modellansicht.

Siehe auch

Erstellen eines Projekts ([siehe Seite 249](#))

Unterstützte Projektformate ([siehe Seite 1332](#))

4.2.3.1.3 Experte zum Konvertieren aus MDL

Dieser Experte unterstützt Sie bei der Erstellung eines Designprojekts aus einem vorhandenen IBM Rational Rose-Modell (MDL). Um den Experten zu starten, wählen Sie im Dialogfeld Neues Projekt die Template **Design-Projekte**▶**Aus MDL konvertieren**. Pfade

Schaltfläche	Beschreibung
Hinzufügen	Mit Hilfe dieser Schaltfläche können Sie dem Bereich <i>Pfade</i> eine Modelldatei hinzufügen. Klicken Sie auf die Schaltfläche, um das Dialogfeld Modelldatei auswählen zu öffnen, navigieren Sie zur gewünschten Modelldatei, und klicken Sie auf Öffnen .
Ordner hinzufügen	Mit dieser Schaltfläche können Sie alle Modelldateien im ausgewählten Ordner hinzufügen. Klicken Sie auf die Schaltfläche, um das Dialogfeld Ordnersuche zu öffnen, navigieren Sie zum Ordner mit den Modelldateien, und klicken Sie auf OK .
Entfernen	Durch Klicken auf diese Schaltfläche löschen Sie den Eintrag, der im Bereich <i>Pfade</i> ausgewählt ist.
Alle entfernen	Wenn Sie auf diese Schaltfläche klicken, werden alle Modelldateien aus dem Bereich <i>Pfade</i> entfernt.

Optionen

Option	Beschreibung
Skalierungsfaktor	Geben Sie in dieses Feld den Koeffizienten für die Elementgröße ein. Die Standardeinstellung für den Skalierungsfaktor ist 0,3.
Rose-Standardfarben konvertieren	Wenn Sie diese Option aktivieren, werden die Farben von Rational Rose durch die Standardfarben von Together ersetzt.
Diagrammknotengrenzen beibehalten	Aktivieren Sie diese Option, wenn benutzerdefinierte Begrenzungen in den resultierenden Diagrammen beibehalten werden sollen. Ist die Option nicht aktiviert, werden die Standardwerte verwendet.

Rose-Akteure konvertieren	Mit dieser Option können Sie den Rose-Klassen Akteur-Stereotypen zuordnen (<i>Actor, Business Actor, Business Worker, Physical Worker</i>). Ist die Option aktiviert, werden den Rose-Akteuren entsprechende Together-Akteure zugeordnet. Wenn Sie die Option nicht aktivieren, werden die Rose-Akteure den Klassen mit dem Stereotyp <i>Actor</i> zugeordnet.
---------------------------	--

Siehe auch

Importieren eines Projekts im IBM Rational Rose-Format (MDL) ([siehe Seite 251](#))

Projekttypen und Formate ([siehe Seite 1332](#))

4.2.3.1.4 Unterstützte Delphi-Projektexperten

Together unterstützt alle Delphi-Projekttypen, die in RAD Studio verfügbar sind.

Diese Projekte können mit einem UML 1.5-Modell ausgestattet werden.

Weitere Informationen über das Erstellen von Delphi-Projekten finden Sie in der Dokumentation von RAD Studio.

Siehe auch

Ein Projekt erstellen ([siehe Seite 249](#))

Unterstützte Projektfunktionen ([siehe Seite 1332](#))

4.2.3.1.5 Unterstützte C# Projektexperten

Together unterstützt alle C# Projekttypen, die in RAD Studio verfügbar sind.

Diese Projekte können mit einem UML 1.5-Modell ausgestattet werden.

Weitere Informationen über das Erstellen von C# Projekten finden Sie in der Dokumentation von RAD Studio.

Siehe auch

Ein Projekt erstellen ([siehe Seite 249](#))

Unterstützte Projektfunktionen ([siehe Seite 1332](#))

4.2.3.2 Pattern-Experte

Der Pattern-Experte unterstützt Sie beim Anwenden eines Pattern. Sie können den Pattern-Experten auf folgende Arten öffnen:

- Klicken Sie in der der Tool-Palette auf die Schaltfläche Knoten nach Pattern oder Beziehung nach Pattern.
- Wählen Sie im Kontextmenü der Diagrammansicht oder einer Klasse den Befehl Nach Pattern erstellen.

Beim Hinzufügen des Elements "Knoten nach Pattern" wird der Pattern-Experte geöffnet. Sie können diesen Experten auch über das Kontextmenü des Diagramms mit dem Befehl Nach Pattern erstellen öffnen.

Element	Beschreibung
Pattern-Hierarchie:	In diesem Feld legen Sie fest, welche Inhalte im Fenster Pattern angezeigt werden. Klicken Sie auf den Abwärtspfeil, um eine Pattern-Hierarchie auszuwählen. Pattern-Hierarchien werden im Pattern-Organizer definiert.
Fenster	Im oberen Bereich des Dialogfeldes befinden sich die folgenden Auswahl-/Editorfenster:
Pattern:	Das Fenster Pattern enthält eine Baumstruktur mit den verfügbaren Pattern. Im Feld Pattern-Hierarchie bestimmen Sie, welche Inhalte in diesem Fenster angezeigt werden.

Pattern-Eigenschaften:	Bearbeiten Sie die generierten Klassennamen für Pattern-Elemente, oder öffnen Sie mit der Informationsschaltfläche das Dialogfeld Element auswählen .
Beschreibung	Im Fenster Beschreibung im unteren Bereich des Dialogfeldes Pattern-Experte wird ein kontextbezogener Hilfetext eingeblendet. Die Beschreibung für komplexere Pattern wird direkt im Pattern-Experten (und nicht in der Online-Hilfe von Together) angezeigt. Sie können die Beschreibung eines Pattern anzeigen, indem Sie im Fenster Pattern auf das Pattern klicken.
Fehler	Falls während der Anwendung eines Pattern ein Fehler auftritt, wird hier die zugehörige Meldung angezeigt (der Pattern-Experte bleibt geöffnet).
Schalter	
OK	Das angegebene Pattern wird angewendet.
Abbrechen	Der Pattern-Experte wird ohne Anwendung eines Pattern geschlossen.

Siehe auch

Modellelemente nach Pattern erstellen (↗ siehe Seite 227)

4.2.3.3 Experte zum Erstellen von Pattern

Mit dem Experten zum Erstellen von Pattern können Sie ein vorhandenes Fragment eines Modells als Pattern speichern und später wieder verwenden. Öffnen Sie den Experten zum Erstellen von Pattern, indem Sie in einem Klassendiagramm einen oder mehrere Knoten und Beziehungen auswählen und im Kontextmenü den Befehl Als Pattern speichern wählen.

Element	Beschreibung
Datei	Dieses Feld gibt den Namen der XML-Zieldatei an.
Name	Der Name des neuen Pattern.
Pattern-Objekte erstellen	Aktivieren Sie dieses Auswahlfeld, damit das Pattern als First-Class-Citizen fungieren kann. Dadurch wird bei jeder Anwendung des Pattern im Diagramm ein ovales Pattern-Element angezeigt.

Siehe auch

Überblick über Pattern (↗ siehe Seite 1454)

Ein Pattern erstellen (↗ siehe Seite 231)

4.2.4 Together-Tastaturkürzel

In Together können viele Diagrammaktionen ohne Zuhilfenahme der Maus über Tastaturkürzel ausgeführt werden. Sie können beispielsweise zwischen Diagrammen wechseln, Diagrammelemente erstellen und viele weitere Operationen durchführen.

Tastaturkürzel für die Navigation

Für die Navigation und Dateisuche stehen folgende Tastaturkürzel zur Verfügung:

Aktion	Kürzel	Bemerkungen
Wechseln zwischen den in der Diagrammansicht geöffneten Diagrammen	STRG+Tab	Der Titel des aktiven Diagramms wird in Fettschrift angezeigt.
Erweitern eines Knotens in der Modellansicht	Rechtspfeil	
Ausblenden eines Knotens in der Modellansicht	Linkspfeil	
Öffnen des Objektinspektors	F4 oder ALT+EINGABE	
Schließen des aktuellen Diagramms	STRG+F4	
Umschalten zwischen einem ausgewählten Container-Knoten und seinen Membern	Bild ab/Bild auf	
Navigation zwischen Knoten oder Knoten-Membern	Pfeiltasten, UMSCHALT+Pfeiltasten	

Tastaturkürzel für die Bearbeitung

Die folgenden Tastaturkürzel für Bearbeitungsaktionen stehen zur Verfügung:

Aktion	Kürzel
Ausschneiden, Kopieren und Einfügen von Modellelementen oder Membern	STRG+X, STRG+C, STRG+V
Aktivieren des internen Editors für ein Diagrammelement zum Bearbeiten oder Umbenennen eines Member	F2
Rückgängig	STRG+Z
Wiederherstellen	STRG+Y, STRG+UMSCHALT+Z
Auswählen aller Elemente im Diagramm	STRG+A
Schließen des Übersichtsfensters	ESC
Hinzufügen eines neuen Namespace (Pakets) zu einem Diagramm	STRG+E
Hinzufügen einer neuen Klasse zu einem Diagramm	STRG+L
Hinzufügen einer neuen Methode (Operation) zu einer Klasse oder einem Interface	STRG+M
Hinzufügen eines neuen Feldes (Attributs) zu einer Klasse	STRG+W
Hinzufügen eines neuen Interface zu einem Diagramm	STRG+UMSCHALT+L
Öffnen des Dialogfelds Verknüpfungen hinzufügen	STRG+UMSCHALT+M
Hinzufügen eines neuen Diagramms aus der Modellansicht	STRG+UMSCHALT+D

Tastaturkürzel für das Zoomen

Folgende Tastaturkürzel für das Zoomen der Diagrammgrafik stehen zur Verfügung:

Aktion	Kürzel	Bemerkungen
Vergrößern	+	Numerischer Tastaturlblock
Verkleinern	-	Numerischer Tastaturlblock
Anzeigen des gesamten Diagramms in der Diagrammansicht	*	Numerischer Tastaturlblock
Anzeigen der tatsächlichen Größe	/	Numerischer Tastaturlblock

Weitere Tastaturkürzel

Folgende zusätzliche Tastaturkürzel stehen zur Verfügung:

Aktion	Kürzel
Öffnen des Dialogfelds Diagramm drucken	STRG+P
Aktualisieren des Diagramms	F6

Siehe auch

[Hilfe zur Hilfe \(siehe Seite 1378\)](#)

[Allgemeines zu Together \(siehe Seite 1380\)](#)

4.2.5 Together-Konfigurationsoptionen

In diesem Abschnitt werden die UML-Modellierungsoptionen beschrieben.

4.2.5.1 Konfigurationsebenen

Die Konfigurationsoptionen wirken sich auf die vier Hierarchieebenen aus. Jede Ebene enthält verschiedene Kategorien (Gruppen) von Optionen.

Die Konfigurationsoptionen können auf folgenden Ebenen festgelegt werden:

- **Vorgabe:** Die Optionen dieser Ebene werden auf alle aktuellen Projekte und Projektgruppen (auch auf neu erstellte) angewendet. Die Diagrammoptionen wirken sich nur auf die neu erstellen Diagramme in diesen Projekten und Projektgruppen aus. Auf dieser Ebene stehen alle Optionen zur Verfügung.
- **Projektgruppe:** Die Optionen dieser Ebene werden auf die aktuelle Projektgruppe angewendet. Die Diagrammoptionen wirken sich nur auf die neu erstellen Diagramme in dieser Projektgruppe aus. Auf dieser Ebene stehen alle Optionen zur Verfügung.
- **Projekt:** Die Optionen dieser Ebene werden auf das aktuelle Projekt angewendet. Die Diagrammoptionen wirken sich nur auf die neu erstellen Diagramme in diesem Projekt aus. Auf dieser Ebene stehen alle Optionen mit Ausnahme der Kategorie **Modellansicht** zur Verfügung.
- **Diagramm:** Die Optionen dieser Ebene werden auf das aktuelle Diagramm angewendet. Auf dieser Ebene steht nur die Kategorie mit den Diagrammoptionen zur Verfügung.

Siehe auch

[Together konfigurieren \(siehe Seite 102\)](#)

[Optionen \(Dialogfeld\) \(siehe Seite 696\)](#)

4.2.5.2 Optionswert-Editoren

Um eine Option zu bearbeiten, klicken Sie in das Wertfeld, um den entsprechenden Werteditor aufzurufen. Es gibt folgende Arten von Werteditoren:

- **Interner Texteditor.** Um den Wert eines Textfelds zu ändern, geben Sie den neuen Wert ein. Die Änderungen werden übernommen, wenn Sie die EINGABETASTE drücken oder zu einem anderen Feld wechseln.
- **Kombinationsfeld oder Listenfeld.** Wenn Sie auf ein Kombinations- bzw. Listenfeld klicken, wird die Liste der möglichen

Werte angezeigt. Wählen Sie dann den gewünschten Wert in der Liste aus.

- **Dialogfeld.** Wenn Sie in ein Feld klicken, das zur Bearbeitung über ein Dialogfeld verfügt, wird die Schaltfläche zum Öffnen des Dialogfelds angezeigt. Geben Sie dann die gewünschten Werte an, und klicken Sie auf **OK**, um die Änderungen zu übernehmen.

Siehe auch

[Konfigurieren von Together \(siehe Seite 102\)](#)

[Optionen \(Dialogfeld\) \(siehe Seite 696\)](#)

4.2.5.3 Together-Optionskategorien

4

In diesem Abschnitt werden die Optionskategorien für die Modellierung beschrieben.

4.2.5.3.1 Together – Allgemeine Optionen

[Tools > Optionen > Together > Verschiedene > Allgemein](#)

Beschreibung der Optionen der Kategorie **Allgemein**.

Mit den Optionen der Kategorie **Allgemein** können Sie bestimmte Verhaltensweisen der Benutzeroberfläche anpassen, die nicht durch eine andere Optionskategorie (etwa die Kategorie für Diagramme oder Ansichtsfilter) abgedeckt werden. Die folgende Tabelle enthält eine Beschreibung dieser Optionen sowie den jeweiligen Vorgabewert.

Allgemeine Optionen	Beschreibung und Standardwert
Löschen bestätigen	Diese Option definiert, ob vor dem Löschen eines Namespace, Klassifizierers oder Diagramms eine Bestätigung erforderlich ist. Der Standardwert ist True .

Together-Unterstützung	Beschreibung und Standardwert
Together-Unterstützung automatisch für neue und geöffnete Projekte aktivieren	Diese Option legt fest, ob die Together-Unterstützung automatisch für geöffnete und neue Projekte aktiviert wird, die zu einer vorhandenen Projektgruppe hinzugefügt werden. Ein Projekt wird als neu betrachtet, wenn es in einer vorhandenen Projektgruppe mit dem Befehl Datei Neu <Projekttyp> erstellt wird. Beachten Sie, dass die im Dialogfeld Modellunterstützung aktivierte Together-Unterstützung Vorrang vor dieser Einstellung hat. Der Standardwert ist True .
	geöffnet werden, die noch nie Gegenstand der Modellierungsunterstützung war. Wenn Sie eine Projektgruppe erstellen, wird ihr Standardprojekt als geöffnet betrachtet. Beachten Sie, dass die im Dialogfeld Modellunterstützung aktivierte Together-Unterstützung Vorrang vor dieser Einstellung hat.

Siehe auch

[Überblick zur Interoperabilität \(siehe Seite 1460\)](#)

[Konfigurieren von Together \(siehe Seite 102\)](#)

[Together-Unterstützung für Projekte aktivieren \(siehe Seite 248\)](#)

4.2.5.3.2 Together-Diagrammoptionen (Erscheinungsbild)

Tools > Optionen > Together > Verschiedene > Diagramm > Erscheinungsbild

Beschreibung der Optionen, mit denen das Erscheinungsbild von Diagrammen beeinflusst werden kann.

Mit den Optionen der Kategorie Diagramm legen Sie Standardverhaltensweisen und das Erscheinungsbild von Diagrammen fest. Die folgende Tabelle enthält eine Beschreibung dieser Optionen sowie den jeweiligen Vorgabewert.

Allgemeine Optionen	Beschreibung und Standardwert
Benutzerdefinierte Hintergrundfarbe des Diagramms	Mit diesem Parameter wird die Hintergrundfarbe von Diagrammen festgelegt, wenn die Option Standardhintergrundfarbe verwenden auf False gesetzt ist. Der Vorgabewert ist WhiteSmoke .
Detailebene des Diagramms	Diese Option definiert, welche Informationen für ein Element in Diagrammen angezeigt werden. Der Standardwert ist Entwurf .
Entwurf	Namen und Typen (Sichtbarkeit wird angezeigt)
Analyse	Nur Namen (keine Sichtbarkeit)
Implementierung	Namen und Typen, Parameter für Methoden und Anfangswerte der Attribute (Sichtbarkeit wird angezeigt)
Schriftart in Diagrammen	Diese Option definiert die Schrift und die Schriftgröße für gedruckte Diagramme. Die Standardschriftart ist Microsoft Sans Serif, 12 Pt.
Textknoten umbrechen	Mit dieser Option steuern Sie, ob der Text in einem Knoten automatisch in die nächste Zeile umbrochen wird, wenn der rechte Rand des Knotens erreicht ist. Bei False wird der Rest des Textes nicht angezeigt. Der Standardwert ist True .
Maximale Breite der Beschriftungen (Pixel)	Mit dieser Einstellung legen Sie die maximale Breite für alle Beschriftungen außerhalb von Knoten fest (z.B. für Beziehungsbeschriftungen). Längerer Beschriftungstext wird automatisch umbrochen. Wenn Sie für diese Einstellung den Wert 0 festlegen, wird der Text immer in einer einzelnen Zeile angezeigt. Der Standardwert ist 200 .
Member-Format	Mit dieser Option wird das Format der Member in Klassendiagrammen gesteuert. Der Standardwert ist UML .
UML	Methoden (Funktionen) werden als <code><Name> (Parameter) : <Typ></code> angezeigt. Felder werden in der Form <code><Name> : <Typ></code> angezeigt.
Sprache	Methoden in C# werden in der Form <code><Typ> <Name></code> angezeigt. Felder in C# werden in der Form <code><Typ> <Name></code> angezeigt.
Seitenränder anzeigen	Diese Option steuert, ob Seitenränder in der Diagrammansicht und in der Übersicht als graue Rahmen angezeigt werden. Der Vorgabewert ist False .
Standardhintergrundfarbe verwenden	Dieser Parameter legt fest, ob in Diagrammen die Systemhintergrundfarbe verwendet wird. Wenn dieser Parameter True ist, wird die Hintergrundfarbe entsprechend dem aktuellen Windows-Farbschema festgelegt. Hat der Parameter den Wert False , wird die Hintergrundfarbe von der Option Benutzerdefinierte Hintergrundfarbe des Diagramms bestimmt (siehe oben). Der Standardwert ist True .

Rasteroptionen	Beschreibung und Standardwert
Rasterfarbe	Dieser Parameter definiert die Farbe des Diagrammrasters. Der Vorgabewert ist LightGray .
Rasterhöhe (Pixel)	Diese Option ermöglicht die Angabe der genauen Höhe von Rasterquadraten in Pixel. Der Standardwert ist 10 .
Rasterstil	Dieser Parameter legt fest, ob für die Darstellung des Rasters gepunktete oder durchgezogenen Linien verwendet werden. Der Vorgabewert ist Linien .
Rasterbreite (Pixel)	Diese Option ermöglicht die Angabe der genauen Breite von Rasterquadraten in Pixel. Der Standardwert ist 10 .
Raster anzeigen	Wenn diese Option True ist, wird im Hintergrund von Diagrammen ein Design-Raster angezeigt. Der Standardwert ist True .
Am Raster ausrichten	Wenn diese Option True ist, werden Diagrammelemente am nächsten Rasterpunkt des Design-Rasters ausgerichtet. Diese Funktion ist wirksam, unabhängig davon, ob das Raster ein- oder ausgeblendet ist. Der Standardwert ist True .

Knotenoptionen	Beschreibung und Standardwert
3D	Wenn diese Option True ist, erscheint unter jedem Diagrammelement ein Schatten, um einen dreidimensionalen Effekt zu erzeugen. Der Standardwert ist True .
Abschnitte als Linie anzeigen	Wenn diese Option True ist, wird über den Abschnitten der markierten Diagrammelemente (Felder, Methoden, Klassen und Eigenschaften) eine Steuerleiste angezeigt. Die Abschnitte werden als erweiterbare Knoten dargestellt, die mit einem Mausklick geöffnet und geschlossen werden können. Der Standardwert ist True .
Importierte Klassen mit voll qualifizierten Namen anzeigen	Dieser Parameter legt fest, ob importierte Klassennamen in der voll qualifizierten oder kurzen Form angezeigt werden. Der Standardwert ist True .
Referenzierte Klassennamen anzeigen	Mit Hilfe dieses Parameters können Sie den Namen der Basisklasse bzw. des Interface in der oberen rechten Ecke eines Klassifizierers in der Diagrammansicht ein- oder ausblenden. Sie können diese Referenzen ausblenden, um die visuelle Darstellung des Projekts übersichtlicher zu gestalten. Der Standardwert ist True .
Referenzierte Klassen mit voll qualifizierten Namen anzeigen	Dieser Parameter legt fest, ob referenzierte Klassennamen in der voll qualifizierten oder kurzen Form angezeigt werden. Der Standardwert ist True .
Elemente nach ihrer Position im Quelltext sortieren	Wenn Sie diese Option auf True setzen, werden Felder, Methoden, Subklassen und Eigenschaften entsprechend ihrer Position im Quelltext sortiert. Diese Option hat Vorrang vor den anderen. Ist sie auf True gesetzt, werden die Optionen Elemente alphabetisch sortieren und Elemente nach Sichtbarkeit sortieren ignoriert.
Elemente alphabetisch sortieren	Diese Option steuert die Reihenfolge der Member, die in Diagrammelementen angezeigt werden. Wenn diese Option True ist, werden Felder, Methoden, Subklassen und Eigenschaften in Bereichen alphabetisch sortiert. Der Standardwert ist True .

Elemente nach Sichtbarkeit sortieren	Diese Option steuert die Reihenfolge der Member, die in Diagrammelementen angezeigt werden. Wenn diese Option True ist, werden Felder, Methoden, Subklassen und Eigenschaften in Bereichen nach Sichtbarkeit sortiert. Der Standardwert ist True .
--------------------------------------	---

Optionen von "UML in Farbe"	Beschreibung und Standardwert
Stereotyp description	Dieser Parameter definiert die Farbe von Klassifizierern mit dem Stereotyp <i>description</i> . Der Standardwert ist LightBlue .
Stereotyp mi-detail	Dieser Parameter definiert die Farbe von Klassifizierern mit dem Stereotyp <i>mi-detail</i> . Der Standardwert ist LightPink .
Stereotyp moment-interval	Dieser Parameter definiert die Farbe von Klassifizierern mit dem Stereotyp <i>moment-interval</i> . Der Standardwert ist LightPink .
Stereotyp party	Dieser Parameter definiert die Farbe von Klassifizierern mit dem Stereotyp <i>party</i> . Der Standardwert ist LightGreen .
Stereotyp place	Dieser Parameter definiert die Farbe von Klassifizierern mit dem Stereotyp <i>place</i> . Der Standardwert ist LightGreen .
Stereotyp role	Dieser Parameter definiert die Farbe von Klassifizierern mit dem Stereotyp <i>role</i> . Der Standardwert ist Yellow .
Stereotyp thing	Dieser Parameter definiert die Farbe von Klassifizierern mit dem Stereotyp <i>thing</i> . Der Standardwert ist LightGreen .
UML in Farbe aktivieren	Diese Option legt fest, ob die Farbe eines Klassifizierers vom zugeordneten Stereotyp abhängig ist. Sie können für jedes Stereotyp die zugehörige Farbe in der Dropdown-Liste auswählen (siehe oben). Der Standardwert ist True .

Siehe auch

Konfigurieren von Together (siehe Seite 102)

Optionen (Dialogfeld) (siehe Seite 696)

4.2.5.3.3 Together-Diagrammoptionen (Layout)

Tools>Optionen>Together>Verschiedene>Layout>Erscheinungsbild

Beschreibung der Optionen, mit denen das Diagramm-Layout beeinflusst werden kann.

Mit den Layout-Optionen können Sie die Ausrichtung der Diagrammelemente steuern.

Allgemeine Optionen	Beschreibung und Standardwert
Layoutalgorithmus	<p>Da man sich UML-Diagramme als Graphen vorstellen kann (mit Eckpunkten und Kanten), kommen bei ihrem Layout auch entsprechende Algorithmen zum Einsatz. Klicken Sie auf den Abwärtspfeil, um einen Layout-Algorithmus auszuwählen. Die verschiedenen Algorithmen und ihre optionale Einstellungen werden im Folgenden beschrieben. Der festgelegte Algorithmus wird ausgeführt, wenn Sie im Kontextmenü des Diagramms auf Layout > Vollständiges Layout klicken. Folgende Optionen stehen zur Wahl:</p> <ul style="list-style-type: none"> · <Auto-Auswahl> · Hierarchisch · Together · Baumartig · Orthogonal · Spring Embedder <p>Der Standardwert ist Together.</p>
Rekursives Layout	<p>Diese Option steht für alle Layout-Algorithmen zur Verfügung. Wenn sie aktiviert ist, werden auch die Unterelemente in den Containerelementen neu angeordnet. Dies ist besonders bei zusammengesetzten Zuständen oder Komponenten hilfreich.</p>

Hierarchischer Algorithmus	Beschreibung und Standardwert
Hybrideverhältnis-Parameter	<p>Diese Einstellung wird in Verbindung mit der Hybride-Heuristik verwendet. Der optimale Wert für diese Einstellung ist 0,7.</p>
Vererbung	<p>Mit dieser Option wird festgelegt, wie Knoten ausgerichtet werden, wenn sie durch eine Vererbungsbeziehung miteinander verbunden sind.</p> <p>Horizontal – Die durch eine Vererbungsbeziehung verbundenen Knoten werden horizontal ausgerichtet.</p> <p>Vertikal – Die durch eine Vererbungsbeziehung verbundenen Knoten werden vertikal ausgerichtet.</p>
Ausrichtung	<p>Mit dieser Option wird die Ausrichtung der Knoten festgelegt. Die Auswirkung der Einstellung hängt von der Option Vererbung ab. Folgende Einstellungen stehen zur Auswahl:</p> <ul style="list-style-type: none"> · Oben: Wenn die Option Vererbung auf Vertikal gesetzt ist, werden alle Knoten in einer Spalte linksbündig ausgerichtet. Hat Vererbung den Wert Horizontal, werden alle Knoten einer Zeile oben in der Zeile ausgerichtet. · Mitte: Wenn die Option Vererbung auf Vertikal gesetzt ist, werden alle Knoten in einer Spalte in der Spaltenmitte ausgerichtet. Hat Vererbung den Wert Horizontal, werden alle Knoten einer Zeile in der Zeilemitte angeordnet. · Unten: Wenn die Option Vererbung auf Vertikal gesetzt ist, werden alle Knoten in einer Spalte rechtsbündig ausgerichtet. Hat Vererbung den Wert Horizontal, werden alle Knoten einer Zeile unten in der Zeile ausgerichtet.
Heuristik zur Einordnung in Schichten	<p>Die Heuristik wird zum Einordnen der Knoten in Schichten verwendet, um Kantenkreuzungen zu minimieren.</p> <p>Schwerpunkt: Diese Heuristik ordnet die Knoten des Knotens N entsprechend der Gewichtung des Schwerpunkts neu an. Die Gewichtung des Knotens N ergibt sich aus dem Durchschnitt der relativen Koordinaten all seiner Nachfolger/Vorgänger.</p> <p>Median: Diese Heuristik ordnet die Knoten des Knotens N entsprechend der Gewichtung der Mittellinie neu an. Die Gewichtung des Knotens N ergibt sich aus dem Durchschnitt seiner relativen Positionen, wobei nur die Koordinaten der beiden zentralen Nachfolger/Vorgänger berücksichtigt werden.</p> <p>Hybride: Diese Heuristik ist eine Kombination von Median und Schwerpunkt.</p>

Minimaler horizontaler/vertikaler Abstand	Der zulässige Minimalabstand zwischen Elementen in Pixel. Hier legen Sie den vertikalen und horizontalen Abstand fest.
---	--

4

Together-Algorithmus	Beschreibung und Standardwert
Vererbung	<p>Mit dieser Option wird festgelegt, wie Knoten ausgerichtet werden, wenn sie durch eine Vererbungsbeziehung miteinander verbunden sind. Sie können zwischen folgenden Einstellungen wählen:</p> <p>Von links nach rechts – Die durch eine Vererbungsbeziehung verbundenen Knoten werden horizontal von links nach rechts ausgerichtet.</p> <p>Von rechts nach links – Die durch eine Vererbungsbeziehung verbundenen Knoten werden horizontal von rechts nach links ausgerichtet.</p> <p>Von oben nach unten – Die durch eine Vererbungsbeziehung verbundenen Knoten werden vertikal von oben nach unten ausgerichtet.</p> <p>Von unten nach oben – Die durch eine Vererbungsbeziehung verbundenen Knoten werden vertikal von unten nach oben ausgerichtet.</p>
Ausrichtung	<p>Mit dieser Option wird die Ausrichtung der Knoten festgelegt. Die Auswirkung der Einstellung hängt von der Option Vererbung ab. Die Ausrichtung der Elemente können Sie der folgenden Tabelle entnehmen:</p> <p>Vererbung: Ausrichtung: Oben: Mitte: Unten:</p> <p>Von links nach rechts Rechts in der Spalte In der Mitte der Spalte Links in der Spalte</p> <p>Von rechts nach links Links in der Spalte In der Mitte der Spalte Rechts in der Spalte</p> <p>Von oben nach unten Unten in der Zeile In der Mitte der Zeile Oben in der Zeile</p> <p>Von unten nach oben Oben in der Zeile In der Mitte der Zeile Unten in der Zeile</p>

Baumalgorithmus	Beschreibung und Standardwert
Hierarchie	<p>Mit dieser Option wird die Hierarchierichtung der Elemente festgelegt.</p> <p>Horizontal – Die Elemente werden horizontal ausgerichtet.</p> <p>Vertikal – Die Elemente werden vertikal ausgerichtet.</p>
Hierarchie umdrehen	Die letzten Elemente der Hierarchie werden als erste im Diagramm dargestellt.
Minimaler horizontaler (vertikaler) Abstand	Der Abstand zwischen den Elementen in Pixel. Hier legen Sie den vertikalen und horizontalen Abstand fest.
Ausrichtung	<p>Mit dieser Option wird die Ausrichtung der Elemente festgelegt. Die Auswirkung der Einstellung hängt von der Option Hierarchierichtung ab. Folgende Einstellungen stehen zur Auswahl:</p> <ul style="list-style-type: none"> Oben: Wenn die Option Hierarchierichtung auf Vertikal gesetzt ist, werden alle Knoten in einer Spalte linksbündig ausgerichtet. Hat Hierarchierichtung den Wert Horizontal, werden alle Knoten einer Zeile oben in der Zeile ausgerichtet. Mitte: Wenn die Option Hierarchierichtung auf Vertikal gesetzt ist, werden alle Knoten in einer Spalte in der Spaltenmitte angeordnet. Hat Hierarchierichtung den Wert Horizontal, werden alle Knoten einer Zeile in der Zeilenmitte angeordnet. Unten: Wenn die Option Hierarchierichtung auf Vertikal gesetzt ist, werden alle Knoten in einer Spalte rechtsbündig ausgerichtet. Hat Hierarchierichtung den Wert Horizontal, werden alle Knoten einer Zeile unten in der Zeile angeordnet.
Nicht-Baumkanten verarbeiten	Wenn diese Option markiert ist, werden Nicht-Baumkanten so gebogen, dass sie in das Diagramm-Layout passen.

Orthogonaler Algorithmus		Beschreibung und Standardwert
Strategie Knotenplatzierung	der	<p>Es gibt drei Strategien für die Platzierung von Knoten: Baumartig, Gleichmäßig und Intelligent.</p> <ul style="list-style-type: none"> Baumartig: Diese Strategie zeigt die Diagramme entsprechend der maximalen Ausdehnung in einer Baumstruktur an. Die Ausdehnung wird berechnet, und die Diagrammknoten werden so platziert, dass die Länge der Baumkanten minimiert wird. Dadurch wird der Abstand zwischen Knoten minimiert, die mit einer Baumkante verbunden sind. Gleichmäßig: Diese Strategie verwendet eine gleichmäßige Anordnung des Graphengitters als Ausgangspunkt. Dabei werden die Nachbarn jedes Gitterpunktes V so gleichmäßig wie möglich links und rechts von V verteilt. Intelligent: Diese Strategie sortiert alle Gitterpunkte nach ihren Ein-/Aus-Graden und füllt das Gitter beginnend im Zentrum. Dabei werden zuerst die Gitterpunkte platziert, die höhere Werte aufweisen.
Abstand zwischen Elementen	zwischen	Der Abstand wird in Pixel angegeben. Geben Sie den Mindestabstand zwischen den Diagrammelementen ein.

Algorithmus Spring Embedder		Beschreibung und Standardwert
Federkraft		Die Festigkeit der Federn. Je höher dieser Wert, desto kürzer die Kanten im Ergebnisgraphen.
Federbewegungsfaktor		Der Bewegungsfaktor der Knoten. Je höher der Wert, desto größer ist der Abstand zwischen den Knoten im fertigen Graphen. Der Wert 0 führt zu einem zufälligen Layout der Knoten.

Siehe auch

Diagramm-Layout – Überblick (siehe Seite 1450)

Konfigurieren von Together (siehe Seite 102)

Diagramm-Layout festlegen (siehe Seite 127)

4.2.5.3.4 Together-Diagrammoptionen (Drucken)

Tools▶ **Optionen**▶ **Together**▶ **Verschiedene**▶ **Diagramm**▶ **Drucken**

Beschreibung der Optionen für das Drucken von Diagrammen.

Mit den Optionen der Kategorie Drucken definieren Sie Standardeinstellungen für den Ausdruck von Diagrammen.

Beachten Sie, dass neben diesen Einstellungen auch betriebssystem- und druckerspezifische Optionen wirksam sind.

Die folgenden Tabellen enthalten eine Beschreibung dieser Optionen sowie den jeweiligen Vorgabewert.

Erscheinungsbild		Beschreibung und Standardwert
Abschnitte als Linie drucken		<p>Wenn diese Option auf True gesetzt ist, wird über Abschnitten mit Diagrammelementen (Felder, Methoden, Klassen, Eigenschaften usw.) eine Steuerleiste angezeigt. Die Abschnittsknoten sind in der Druckausgabe erweitert.</p> <p>Der Vorgabewert ist False.</p>
Schatten drucken		<p>Wenn diese Option auf True gesetzt ist, erscheint unter jedem Diagrammelement im Ausdruck ein Schatten, um einen dreidimensionalen Effekt zu erzeugen.</p> <p>Der Standardwert ist True.</p>

Fußzeile	Beschreibung und Standardwert
Fußzeilenausrichtung	Diese Option ermöglicht die Auswahl der Ausrichtung der Fußzeile. Der Standardwert ist Links .
Fußzeilentext	Mit dieser Option wird der Text festgelegt, der am unteren Rand jeder Seite gedruckt werden soll, wenn die Option Fußzeile drucken auf True gesetzt ist. Im Text können Systemmakros verwendet werden. Weitere Informationen finden Sie unter Systemmakros (siehe Seite 681). Der Standardwert ist <code>Printed by %USER% (%LONGDATE%)</code> .
Fußzeile drucken	Wenn diese Option True ist, wird der bei der Option Fußzeilentext angegebene Text am unteren Rand jeder Seite gedruckt. Der Standardwert ist True .

Allgemeine Druckoptionen	Beschreibung und Standardwert
An Seitengröße anpassen	Wenn diese Option True ist, wird die Einstellung des Druck-Zooms ignoriert und das gesamte Diagramm auf eine einzelne Seite gedruckt. Der Vorgabewert ist False .
Schriftart	Diese Option definiert die Schrift und die Schriftgröße für gedruckte Diagramme. Der Standardwert ist <code>Microsoft Sans Serif, 8 Pt.</code>
Rahmen drucken	Wenn diese Option True ist, wird ein Rahmen um jede Seite gedruckt. Der Rahmen entspricht den Rändereinstellungen. Der Standardwert ist True .
Leere Seiten drucken	Wenn diese Option True ist, werden auch leere Seiten gedruckt. Ist diese Option False , werden leere Seiten beim Drucken weggelassen. Der Vorgabewert ist False .
Druck-Zoom	Diese Option legt den Zoom-Faktor für den Druck von Diagrammen fest (1:1, 2:1 usw.). Der Wert 1 entspricht 100 %. Der Wert 2 entspricht dem Zoom-Faktor 200 %, der Wert 0,5 dem Zoom-Faktor 50 % usw. Verwenden Sie das in den lokalen Einstellungen definierte Dezimalzeichen. Der Standardwert ist 1 .

Kopfzeile	Beschreibung und Standardwert
Kopfzeilenausrichtung	Diese Option ermöglicht die Auswahl einer Ausrichtung für die Kopfzeile. Der Standardwert ist Links .
Kopfzeilentext	Diese Option definiert den Text, der oben auf jeder Seite gedruckt werden soll, wenn die Option Kopfzeile drucken auf True gesetzt ist. Im Text können Systemmakros verwendet werden. Weitere Informationen finden Sie unter Systemmakros (siehe Seite 681). Der Standardwert ist <code>%PROJECT%, %DIAGRAM%</code> .
Kopfzeile drucken	Wenn diese Option True ist, wird der in der Option Kopfzeilentext angegebene Text am oberen Rand jeder Seite gedruckt. Der Standardwert ist True .

Ränder	Beschreibung und Standardwert
Unterer Rand	Der untere Rand in Zoll. Verwenden Sie das in den lokalen Einstellungen definierte Dezimalzeichen. Der Standardwert ist 1 .
Linker Rand	Der linke Rand in Zoll. Verwenden Sie das in den lokalen Einstellungen definierte Dezimalzeichen. Der Standardwert ist 1 .
Rechter Rand	Der rechte Rand in Zoll. Verwenden Sie das in den lokalen Einstellungen definierte Dezimalzeichen. Der Standardwert ist 1 .
Oberer Rand	Der obere Rand in Zoll. Verwenden Sie das in den lokalen Einstellungen definierte Dezimalzeichen. Der Standardwert ist 1 .

Seitenzahlen	Beschreibung und Standardwert
Ausrichtung Seitenzahlen	der Diese Option ermöglicht die Auswahl einer Ausrichtung für die Seitenzahl. Der Standardwert ist Links .
Seitenzahlen drucken	Wenn diese Option True ist, werden Seitenzahlen gedruckt. Der Standardwert ist True .

Papier	Beschreibung und Standardwert
Benutzerdefinierte Papierhöhe (Zoll)	Diese Option definiert benutzerdefinierte Papierabmessungen für das Drucken. Diese Einstellungen wirken sich nur aus, wenn Papiergröße auf Individuell eingestellt ist. Der Standardwert ist 11,88 .
Benutzerdefinierte Papierbreite (Zoll)	Diese Option definiert benutzerdefinierte Papierabmessungen für das Drucken. Diese Einstellungen wirken sich nur aus, wenn Papiergröße auf Individuell eingestellt ist. Der Standardwert ist 8,4 .
Papierausrichtung	Diese Option definiert die Ausrichtung der Seite. Wenn für die Papiergröße Individuell gewählt ist, werden für die Ausrichtung Hochformat die Werte der Optionen Benutzerdefinierte Papierhöhe und Benutzerdefinierte Papierbreite verwendet. Für die Ausrichtung Querformat werden diese Werte vertauscht. Der Standardwert ist Hochformat .
Papiergröße	Diese Option definiert die Standardpapiergröße für das Drucken. Zu den Auswahlmöglichkeiten gehören die gebräuchlichsten Papiergrößen. Wenn Sie eigene Größen definieren möchten, wählen Sie Individuell und legen die Abmessungen in den Feldern Benutzerdefinierte Papierhöhe und Benutzerdefinierte Papierbreite fest. Der Standardwert ist A4 .

Siehe auch

Konfigurieren von Together (☞ siehe Seite 102)

Optionen (Dialogfeld) (☞ siehe Seite 696)

4.2.5.3.5 Together-Diagrammoptionen (Ansichtsverwaltung)

Tools▶**Optionen**▶**Together**▶**Verschiedene**▶**Diagramm**▶**Ansichtsverwaltung**

Beschreibung der Ansichtsfilteroptionen.

Die Optionsgruppe **Ansichtsverwaltung** enthält Filter, mit denen Sie steuern können, welche Datentypen in den verschiedenen Ansichten eines Modells angezeigt werden.

Mit den Optionen für Ansichtsfilter wird festgelegt, welche Elemente in Klassen- und Namespace-Diagrammen (Paketdiagrammen) angezeigt werden. Die folgende Tabelle enthält eine Beschreibung der Filter sowie den jeweiligen Vorgabewert.

Beziehungsfilter	Beschreibung und Standardwert
Assoziationsbeziehungen anzeigen	Diese Filteroption steuert das Ein-/Ausblenden von Assoziationsbeziehungen im aktuellen Projekt. Wenn die Option auf True gesetzt ist, werden Assoziationsbeziehungen angezeigt. Der Standardwert ist True .
Abhängigkeitsbeziehungen anzeigen	Diese Filteroption steuert das Ein-/Ausblenden von Abhängigkeitsbeziehungen im aktuellen Projekt. Wenn die Option auf True gesetzt ist, werden Abhängigkeitsbeziehungen angezeigt. Der Standardwert ist True .
Generalisierungsbeziehungen anzeigen	Diese Filteroption steuert das Ein-/Ausblenden von Generalisierungsbeziehungen im aktuellen Projekt. Wenn die Option auf True gesetzt ist, werden Generalisierungsbeziehungen angezeigt. Der Standardwert ist True .
Implementierungsbeziehungen anzeigen	Diese Filteroption steuert das Ein-/Ausblenden von Implementierungsbeziehungen im aktuellen Projekt. Wenn diese Option auf True gesetzt ist, werden die Implementierungsbeziehungen angezeigt. Der Standardwert ist True .

Member-Filter	Beschreibung und Standardwert
Felder anzeigen	Diese Filteroption steuert das Ein-/Ausblenden von Feldern im aktuellen Projekt. Wenn die Option auf True gesetzt ist, werden die Felder angezeigt. Der Standardwert ist True .
Indizierer anzeigen	Diese Filteroption steuert das Ein-/Ausblenden von Indizierern im aktuellen Projekt. Wenn die Option auf True gesetzt ist, werden Indizierer angezeigt. Der Standardwert ist True .
Member anzeigen	Diese Filteroption steuert das Ein-/Ausblenden der Member im aktuellen Projekt. Wenn die Option auf True gesetzt ist, werden die Member angezeigt. Der Standardwert ist True .
Methoden anzeigen	Diese Filteroption steuert das Ein-/Ausblenden von Methoden im aktuellen Projekt. Wenn die Option auf True gesetzt ist, werden Methoden angezeigt. Der Standardwert ist True .
Nicht-public Member anzeigen	Diese Filteroption steuert das Ein-/Ausblenden der nicht-public Member im aktuellen Projekt. Wenn die Option auf True gesetzt ist, werden nicht-public Member angezeigt. Der Standardwert ist True .
Eigenschaften anzeigen	Diese Filteroption steuert das Ein-/Ausblenden von Eigenschaften im aktuellen Projekt. Wenn die Option auf True gesetzt ist, werden die Eigenschaften angezeigt. Der Standardwert ist True .

Knotenfilter		Beschreibung und Standardwert
Klassen anzeigen		Diese Filteroption steuert das Ein-/Ausblenden von Klassen im aktuellen Projekt. Wenn die Option auf True gesetzt ist, werden Klassen angezeigt. Der Standardwert ist True .
Einschränkungen anzeigen		Diese Filteroption steuert das Ein-/Ausblenden von OCL-Einschränkungen im aktuellen Projekt. Wenn die Option auf True gesetzt ist, werden die Einschränkungen angezeigt. Der Standardwert ist True .
Delegaten anzeigen		Diese Filteroption steuert das Ein-/Ausblenden von Delegaten im aktuellen Projekt. Wenn die Option auf True gesetzt ist, werden Delegaten angezeigt. Der Standardwert ist True .
Aufzählungen anzeigen		Diese Filteroption steuert das Ein-/Ausblenden von Aufzählungen in dem aktuellen Projekt. Wenn die Option auf True gesetzt ist, werden Aufzählungen angezeigt. Der Standardwert ist True .
Ereignisse anzeigen		Diese Filteroption steuert das Ein-/Ausblenden von Ereignissen im aktuellen Projekt. Wenn die Option auf True gesetzt ist, werden Ereignisse angezeigt. Der Standardwert ist True .
Interfaces anzeigen		Diese Filteroption steuert das Ein-/Ausblenden von Interfaces in dem aktuellen Projekt. Wenn die Option auf True gesetzt ist, werden Interfaces angezeigt. Der Standardwert ist True .
Module anzeigen		Diese Filteroption steuert das Ein-/Ausblenden von Modulen im aktuellen Projekt. Wenn die Option auf True gesetzt ist, werden Module angezeigt. Der Standardwert ist True .
Namespaces anzeigen		Diese Filteroption steuert das Ein-/Ausblenden von Namespaces im aktuellen Projekt. Wenn die Option auf True gesetzt ist, werden Namespaces angezeigt. Der Standardwert ist True .
Nicht-public Klassen anzeigen		Diese Filteroption steuert das Ein-/Ausblenden von nicht-public Klassen im aktuellen Projekt. Wenn die Option auf True gesetzt ist, werden nicht-public Klassen angezeigt. Der Standardwert ist True .
Hinweise anzeigen		Diese Filteroption steuert das Ein-/Ausblenden von Hinweisen im aktuellen Projekt. Wenn die Option auf True gesetzt ist, werden Hinweise angezeigt. Der Standardwert ist True .
Verknüpfungen anzeigen		Diese Filteroption steuert das Ein-/Ausblenden von Verknüpfungen im aktuellen Projekt. Wenn diese Option auf True gesetzt ist, werden Verknüpfungen angezeigt. Der Standardwert ist True .
Strukturen anzeigen		Diese Filteroption steuert das Ein-/Ausblenden von Strukturen im aktuellen Projekt. Wenn die Option auf True gesetzt ist, werden Strukturen angezeigt. Der Standardwert ist True .

Siehe auch

Konfigurieren von Together (☞ siehe Seite 102)

Optionen (Dialogfeld) (☞ siehe Seite 696)

4.2.5.3.6 Together-Optionen für die Dokumentationserzeugung

Tools ▶ Optionen ▶ Together ▶ Verschiedene ▶ Dokumentation generieren

Beschreibung der Optionen für die Dokumentationserzeugung.

Mit den Optionen der Kategorie **Dokumentation generieren** wird der Inhalt (und dessen Darstellung) festgelegt, der in die erzeugte HTML-Dokumentation aufgenommen werden soll. Die folgende Tabelle enthält eine Beschreibung dieser Optionen sowie den jeweiligen Vorgabewert.

Allgemeine Optionen	Beschreibung und Standardwert	
Unten	Der Text, der unten in jede Ausgabedatei platziert werden soll. Der Text wird am unteren Rand der Seite unter der unteren Navigationsleiste platziert. Er kann HTML-Tags und Whitespaces enthalten.	
Titel der Dokumentation	Der Titel, der oben in der Übersichtsdatei angezeigt werden soll. Der Titel wird zentriert als Überschrift der Ebene 1 direkt unter der oberen Navigationsleiste angezeigt. Die Überschrift kann HTML-Tags und Whitespaces enthalten.	
Fußzeile	Der Fußzeilentext, der unten in jede Ausgabedatei platziert werden soll. Die Fußzeile wird rechts neben der unteren Navigationsleiste angezeigt. Die Überschrift kann HTML-Tags und Whitespaces enthalten.	
Kopfzeile	Der Kopfzeilentext, der oben in jede Ausgabedatei platziert werden soll. Die Kopfzeile wird rechts neben der oberen Navigationsleiste angezeigt. Die Überschrift kann HTML-Tags und Whitespaces enthalten.	
Navigationstyp	<p>Die Position für Beschreibungen von Designelementen:</p> <p>Package: In Package-Dateien.</p> <p>Diagramm: In Diagrammdateien.</p> <p>Der Standardwert ist Package.</p>	
Internen verwenden	Browser	<p>Wenn diese Option True ist, wird zur Darstellung der Dokumentation ein interner Browser verwendet.</p> <p>Der Vorgabewert ist False.</p>
Fenstertitel	Der Titel, der in das HTML-Tag <title> aufgenommen werden soll. Dieser Text wird im Fenstertitel und in allen Browser-Lesezeichen (Favoriten) angezeigt, die für diese Seite angelegt werden. Der Titel darf keine HTML-Tags enthalten, da der Browser diese nicht korrekt interpretieren kann.	

Einbeziehen	Beschreibung und Standardwert	
internal (Package)	<p>Wenn diese Option True ist, werden interne Klassen, Interfaces und Member in der generierten Dokumentation angezeigt.</p> <p>Der Standardwert ist True.</p>	
private	<p>Wenn diese Option True ist, werden in der erzeugten Dokumentation private-Klassen, -Interfaces und -Member angezeigt.</p> <p>Der Vorgabewert ist False.</p>	
protected	<p>Wenn diese Option True ist, werden in der erzeugten Dokumentation protected-Klassen, -Interfaces und -Member angezeigt.</p> <p>Der Vorgabewert ist False.</p>	
protected internal	<p>Wenn diese Option True ist, werden interne Klassen, Interfaces und Member in der generierten Dokumentation angezeigt.</p> <p>Der Vorgabewert ist False.</p>	
public	<p>Wenn diese Option True ist, werden in der erzeugten Dokumentation public-Klassen, -Interfaces und -Member angezeigt.</p> <p>Der Standardwert ist True.</p>	

Navigation	Beschreibung und Standardwert
Hilfe erzeugen	Diese Option steuert, ob der Link HILFE in die Navigationsleisten oben und unten auf allen Ausgabeseiten aufgenommen werden soll. Der Standardwert ist True .
Index erzeugen	Diese Option steuert, ob ein Index erzeugt werden soll. Der Standardwert ist True .
Navigationsleiste erzeugen	Diese Option steuert, ob Navigationsleisten, Kopf- und Fußzeilen oben und unten auf allen erzeugten Seiten angezeigt werden. Der Standardwert ist True .
Hierarchie erzeugen	Diese Option steuert, ob eine Klassen/Interface-Hierarchie erzeugt werden soll. Der Standardwert ist True .
Verwendungsseite erzeugen	Wenn diese Option True ist, wird für alle dokumentierten Klassen und Namespaces eine Verwendungsseite erzeugt. Die Seite beschreibt, welche Namespaces, Klassen, Methoden, Konstruktoren und Felder APIs der gegebenen Klasse oder des Namespace verwenden. Bei gegebener Klasse C würden in die Klasse C alle Subklassen von C, Felder, die als C deklariert sind, Methoden, die C zurückgeben, und Methoden und Konstruktoren mit Parametern des Typs C eingeschlossen sein. Der Vorgabewert ist False .

Siehe auch

Konfigurieren von Together (↗ siehe Seite 102)

Optionen (Dialogfeld) (↗ siehe Seite 696)

4.2.5.3.7 Together-Optionen für die Modellansicht

Tools▶**Optionen**▶**Together**▶**Verschiedene**▶**Modellansicht**

Beschreibung der Optionen für die Modellansicht.

Mit den Optionen der Kategorie **Modellansicht** steuern Sie, wie der Diagramminhalt in der Modellansicht dargestellt wird. Die folgende Tabelle enthält eine Beschreibung dieser Optionen sowie den jeweiligen Vorgabewert.

Option	Beschreibung und Standardwert
Diagrammknoten erweiterbar anzeigen	Wenn diese Option auf True gesetzt ist, werden in der Modellansicht erweiterbare Diagrammknoten mit den enthaltenen Elementen angezeigt. Der Standardwert ist True .
Beziehungen anzeigen	Wenn diese Option auf True gesetzt ist, werden in der Modellansicht Beziehungen zwischen Knoten angezeigt. Andernfalls werden die Beziehungen ausgeblendet. Der Vorgabewert ist False .
Sortiertyp	Diese Option ermöglicht die Auswahl eines Sortiertyps für die Modellansicht (Alphabetisch, Metaklasse oder Ohne). Der Standardwert ist Metaklasse .
Ansichtstyp	Diese Option legt den Typ der Modellpräsentation fest. Im diagrammzentrischen Modus wird davon ausgegangen, dass die Designelemente in den zugehörigen Diagrammen dargestellt und nur die Klassen, Interfaces und anderen Quelltextelemente in den Namespaces angezeigt werden. Im modellzentrischen Modus werden alle Elemente unter den Namespaces angezeigt. Der Standardwert ist Diagrammzentrisch .

Siehe auch

Konfigurieren von Together ( siehe Seite 102)

Optionen (Dialogfeld) ( siehe Seite 696)

4.2.5.3.8 Together-Quelltextoptionen

Tools  **Optionen**  **Together**  **Verschiedene**  **Quelltext**

Beschreibung der Quelltextoptionen.

Mit den Optionen der Kategorie **Quelltext** steuern Sie, ob Abhängigkeitsbeziehungen in Diagrammen automatisch gezeichnet werden. Die folgende Tabelle enthält eine Beschreibung dieser Optionen sowie den jeweiligen Vorgabewert.

Option	Beschreibung und Standardwert
Von Eigenschaften abgeleitete Assoziationsbeziehungen automatisch erzeugen	Setzen Sie diese Option auf True , wenn während der Analyse des Quelltexts automatisch Assoziationsbeziehungen für Eigenschaften erzeugt werden sollen. Der Vorgabewert ist False .
Abhängigkeitsbeziehungen automatisch erstellen	Wenn diese Option True ist, werden die Abhängigkeitsbeziehungen automatisch in Diagrammen gezeichnet. Der Vorgabewert ist False .
Ordnerstruktur Namespaces automatisch beibehalten	Wenn dieser Parameter True ist, werden neue Klassen in Unterordnern erzeugt. Die Struktur dieser Unterordner wird anhand der vorhandenen Ordnerstruktur (falls vorhanden) erkannt. Für neue Namespaces werden neue Unterordner erstellt. Wenn dieser Parameter False ist, werden neue Klassen im Stammdordner des Projekts erzeugt. Der Standardwert ist True .
Codierung	Dieser Parameter legt den Zeichensatz fest, der für Quelltextdateien verwendet werden soll. Bei Auswahl von Systemstandard können Sie den Zeichensatz des aktuellen Betriebssystems verwenden. Voreingestellt ist Systemstandard .

Siehe auch

Konfigurieren von Together ( siehe Seite 102)

4.2.6 UML 1.5-Referenz

Dieser Abschnitt enthält Referenzmaterial für UML 1.5-Diagramme.

4.2.6.1 UML 1.5-Klassendiagramme

Dieser Abschnitt beschreibt die Elemente eines UML 1.5-Klassendiagramms.

4.2.6.1.1 Definition eines UML 1.5-Klassendiagramms

Sie können in Together sprachneutrale Klassendiagramme in Designprojekten oder sprachspezifische Klassendiagramme in

Implementierungsprojekten erstellen. Bei Implementierungsprojekten werden sämtliche Diagrammelemente sofort mit dem Quelltext synchronisiert.

Definition

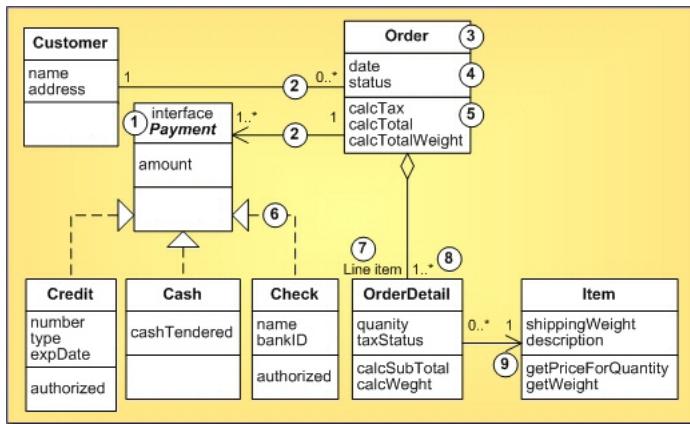
Ein Klassendiagramm liefert einen Überblick über die verschiedenen Klassen eines Systems und ihre Beziehungen untereinander. Klassendiagramme sind statisch: sie zeigen lediglich an, welche Elemente interagieren, aber nicht was dabei geschieht.

In der UML-Notation wird eine Klasse durch ein Rechteck mit drei Abschnitten dargestellt: Klassenname, Felder und Methoden. Die Namen der abstrakten Klassen und der Schnittstellen werden kursiv geschrieben. Die Beziehungen zwischen den Klassen werden durch Verbindungslien angezeigt.

In Together ist das Rechteck weiter unterteilt und enthält separate Abschnitte für Eigenschaften und innere Klassen.

Beispieldiagramm

Das folgende Klassendiagramm modelliert die Bestellung eines Kunden in einem Katalog. Die zentrale Klasse ist **Order**. Mit ihr verknüpft sind die Komponenten **Customer** (Bestellung) und **Payment** (Bezahlung). Es gibt drei Arten von Zahlungen: **Cash**, **Check** und **Credit**. Der Auftrag enthält **OrderDetails** (Posten) mit jeweils einer **Item**-Komponente (Artikel).



① Interface ③ Klassenname ⑤ Methoden ⑦ Rollenname ⑨ Navigierbarkeit
 ② Assoziation ④ Felder ⑥ Implementierung ⑧ Multiplizität

In diesem Beispiel werden drei Arten von Beziehungen verwendet:

- Assoziation: Beispielsweise repräsentiert **OrderDetail** einen Artikel in jeder **Order**-Komponente.
- Aggregation: Im Beispieldiagramm besitzt **Order** eine Sammlung von **OrderDetails**.
- Implementierung: Im Beispieldiagramm ist **Payment** ein Interface für **Cash**, **Check** und **Credit**.

4.2.6.1.2 Arten von Klassendiagrammen

In Together gibt es zwei Arten von Klassendiagrammen:

- **Paketdiagramme** (Namespace-Diagramme). In Designprojekten wird der Begriff **Paketdiagramm**, in Implementierungsprojekten der Begriff **Namespace-Diagramm** verwendet. Diese Diagramme werden als XML-Dateien mit der Namenserweiterung **.txvpck** im Ordner **ModelSupport_%PROJECTNAME%ModelSupport** der Projektgruppe gespeichert.
- **Logische Klassendiagramme**. Diese Diagramme werden als XML-Dateien mit der Namenserweiterung **.txvccls** im Ordner **ModelSupport_%PROJECTNAME%ModelSupport** der Projektgruppe gespeichert.

Together erstellt ein Namespace-Standarddiagramm für das Projekt und für jedes Unterverzeichnis im Projektverzeichnis. Das Standarddiagramm für das Projekt erhält den Namen **default**, die Diagramme für die untergeordneten Verzeichnisse werden nach den jeweiligen Namespaces (Paketen) benannt.

Sie müssen logische Klassendiagramme manuell über das Kontextmenü des Projekts mit dem Befehl **Hinzufügen > Klassendiagramm** oder **Hinzufügen > Anderes Diagramm** erstellen.

Siehe auch

UML 1.5-Klassendiagramm (siehe Seite 1294)

UML 2.0-Klassendiagramm (siehe Seite 1313)

4.2.6.1.3 Elemente in einem UML 1.5-Klassendiagramm

In der folgenden Tabelle sind die Elemente eines UML 1.5-Klassendiagramms aufgeführt, die in der Tool Palette zur Verfügung stehen.

Elemente in einem UML 1.5-Klassendiagramm

Name	Typ
Namespace (Paket)	Knoten
Klasse	Knoten
Interface	Knoten
Assoziationsklasse	Knoten
Struktur	Knoten
Aufzählung	Knoten
Delegat	Knoten
Objekt	Knoten
Generalisierung/Implementierung	Beziehung
Assoziation	Beziehung
Abhängigkeit	Beziehung
Knoten nach Pattern	Öffnet den Pattern-Experten (siehe Seite 1277).
Beziehung nach Pattern	Öffnet den Pattern-Experten (siehe Seite 1277).
Constraint	OCL-Knoten
Einschränkungsbeziehung	OCL-Beziehung
Hinweis	Annotation
Hinweisbeziehung	Annotationsbeziehung

4.2.6.1.4 Beziehungen in Klassendiagrammen

Es gibt mehrere Arten von Beziehungen:

- **Assoziation:** Eine Beziehung zwischen Instanzen zweier Klassen. Assoziationen zwischen zwei Klassen bestehen, wenn eine Instanz der einen Klasse von der anderen wissen muss, um ihre Aufgaben durchzuführen. In einem Diagramm wird eine Assoziation durch eine Beziehungslinie zwischen zwei Klassen dargestellt. Assoziationen können gerichtet oder ungerichtet sein. Eine gerichtete Beziehung zeigt auf die Anbieterklasse (das Ziel). Eine Assoziation besitzt zwei Enden. Ein Ende kann mit einem Rollennamen versehen sein, der den Charakter der Assoziation beschreibt. Ein Navigationspfeil auf einer Assoziation zeigt die Richtung an, in der die Assoziation "durchquert" (abgefragt) werden kann. Für eine Klasse kann eine Abfrage zu ihrem Element gestellt werden. Abfragen in entgegengesetzter Richtung sind nicht möglich. Am Pfeil lässt sich auch erkennen, wer der "Eigentümer" der Implementierung der Assoziation ist. Assoziationen ohne Navigationspfeil sind bidirektional.

- **Generalisierung/Implementierung:** Eine Vererbungsbeziehung, die anzeigt, dass eine Klasse ein Interface implementiert. Eine Implementierung wird mit einem Dreieck dargestellt, das auf das Interface zeigt.
- **Abhangigkeit**

Es gibt verschiedene Untertypen von Assoziationsbeziehungen:

- **Einfache Assoziation**

- **Aggregation:** Eine Assoziation, in der eine Klasse zu einer Sammlung gehort. Eine Aggregationsbeziehung wird mit einem rautenformigen Ende dargestellt, das auf den Container zeigt.

- **Komposition**

Jedes Klassendiagramm enthalt Klassen und Assoziationen. Navigationsmglichkeiten, Rollen und Multiplizitat sind optionale Elemente, die die Interpretation der Diagramms vereinfachen.

Die Multiplizitat eines Assoziationsendes bestimmt, wie viele Instanzen der Klasse mit einer einzelnen Instanz am anderen Ende verbunden sein knnen. Die Multiplizitat kann als Einzelwert oder als Wertebereich angegeben werden. In der folgenden Tabelle sind die gangigsten Arten der Multiplizitat aufgefuhrt:

Multiplizitat

Multiplizitat	Bedeutung
0..1	Keine oder eine Instanz. Die Notation $n \dots m$ steht fur n bis m Instanzen.
0..* oder *	Keine Beschrnkung bezuglich der Zahl der Instanzen (auch keine Instanz ist mglich).
1	Genau eine Instanz.
1..*	Mindestens eine Instanz.

Siehe auch

UML 1.5-Klassendiagramm ([siehe Seite 1294](#))

UML 2.0-Klassendiagramm ([siehe Seite 1313](#))

4.2.6.1.5 LiveSource-Regeln

Die Auswirkungen von nderungen einer Klasse, eines Interface oder eines Namespace in einem logischen Klassendiagramm hangen von der jeweiligen Aktion ab:

- Wenn Sie den Namen ndern, ein Member hinzufugen, eine Beziehung erstellen oder ein Pattern anwenden, wird der Quelltext entsprechend gendert.
- Mit dem Befehl Aus Ansicht lschen im Kontextmenu eines Elements knnen Sie dieses aus dem aktuellen Diagramm entfernen, es aber im Namespace (Paket) beibehalten.
- Um ein Element vollstandig aus dem Modell zu entfernen, whlen Sie Lschen im Kontextmenu des Elements.
- Wenn Sie auf der Tastatur **ENTF** drcken, wird der Befehl Aus Ansicht lschen ausgefuhrt, falls er verfigbar ist. Falls nicht, wird das Element vollstandig gelscht.
- Direkte nderungen im Quelltext-Editor, z.B. das Umbenennen einer Klasse, knnen von Together nicht umgesetzt werden. Verwenden Sie fr diesen Zweck die entsprechende Refactoring-Operation.

Siehe auch

LiveSource – bersicht ([siehe Seite 1451](#))

UML 1.5-Klassendiagramm – Referenz ([siehe Seite 1294](#))

4.2.6.1.6 Assoziationsklassen und n-fache Assoziationen

Assoziationsklassen werden in Diagrammen durch drei Elemente dargestellt:

- Assoziationsklasse (Klassensymbol)
- N-fache Assoziationsklassenbeziehung (Raute)
- Assoziationskonnektor (Verbindungslinie zwischen beiden)

Die Assoziationsklassen können Verbindungen mit beliebig vielen Assoziationsendeklassen (Teilnehmern) herstellen.

Im Objektinspektor für Assoziationsklasse, Assoziationsbeziehung und Konnektor wird zusätzlich die Registerkarte **Assoziation** angezeigt. Sie enthält lediglich das Feld **label**, in das automatisch der Name der Assoziationsklasse eingetragen wird. Bei Assoziationsklassen und Assoziationsbeziehungsenden enthält der Knoten **Benutzereigenschaften** des Objektinspektors weitere Eigenschaften, die der Rolle dieses Bestandteils der n-fachen Assoziation entsprechen (`associationClass` bzw. `associationEnd`).

Sie können beliebige Assoziationsbeziehungsenden oder Teilnehmerklassen löschen, ohne dass die gesamte n-fache Assoziation entfernt wird. Wenn Sie jedoch die Assoziationsklasse löschen, werden alle Bestandteile der n-fachen Assoziation entfernt.

Siehe auch

Assoziationsklasse erstellen (siehe Seite 208)

Beziehungen in Klassendiagrammen (siehe Seite 1296)

UML 2.0-Klassendiagramm (siehe Seite 1313)

UML 1.5-Klassendiagramm (siehe Seite 1294)

4.2.6.1.7 Innere Klassifizierer

In der folgenden Tabelle sind die inneren Klassifizierer aufgeführt, die Sie den verschiedenen Container-Elementen hinzufügen können.

Innere Klassifizierer

Container-Element	Verfügbare innere Klassifizierer
Klasse	Klasse Interface Struktur [C#] Delegat [C#] Delegat als Funktion Enum [C#]
Interface	Klasse] Interface Delegat Delegat als Funktion Enum

Struktur	Klasse Interface Struktur Delegat Delegat als Funktion Enum
Modul	Klasse Interface Struktur Delegat Delegat als Funktion Enum

Siehe auch

UML 1.5-Klassendiagramm – Referenz (siehe Seite 1294)

UML 2.0-Klassendiagramm – Referenz (siehe Seite 1313)

4.2.6.1.8 Member

Beachten Sie, dass sich die verfügbaren Member bei Design- und Implementierungsprojekten unterscheiden.

In der folgende Tabelle sind die Member aufgeführt, die den verschiedenen Container-Elementen über das Kontextmenü hinzugefügt werden können. Die jeweilige Projektart ist in eckigen Klammern angegeben.

Verfügbare Member

Container-Element	Verfügbare Member
Klasse	Methode [C#] Operation [Design] Konstruktor [Design, C#] Destruktor [C#] Feld [C#] Attribut [Design] Eigenschaft [C#] Indizierer [C#] Ereignis [C#]
Interface	Methode [C#] Eigenschaft [C#] Indizierer [C#] Ereignis [C#] Attribut [Design] Operation [Design]

4

Struktur	Methode Konstruktor Feld Eigenschaft Indizierer Ereignis
Modul	Funktion Subroutine Konstruktor Feld Eigenschaft
Aufzählung	Enum-Wert

Hinweis zu Implementierungsprojekten: Wenn Sie die Eigenschaft *abstract* einer Methode, einer Eigenschaft oder eines Indizierers (in abstrakten Klassen) im Eigenschaftenfenster auf **True** setzen, wird der Methodenrumpf aus dem Quelltext entfernt. Dies ist das erwartete Verhalten. Wenn Sie die Eigenschaft danach im Objektinspektor wieder auf **False** setzen, wird ein neuer leerer Methodenrumpf eingefügt.

Siehe auch

Einem Container ein Member hinzufügen (↗ siehe Seite 212)

UML 1.5-Klassendiagramm (↗ siehe Seite 1294)

UML 2.0-Klassendiagramm (↗ siehe Seite 1313)

4.2.6.2 UML 1.5-Anwendungsfalldiagramm

Dieser Abschnitt beschreibt die Elemente eines UML 1.5-Anwendungsfalldiagramms.

4.2.6.2.1 Definition des UML 1.5-Anwendungsfalldiagramms

Anwendungsfalldiagramme sind für drei Zwecke hilfreich:

- Bestimmung der Funktionen (Anforderungen): Neue Anwendungsfälle führen oft zu neuen Anforderungen, da das System analysiert wird und der Entwurf Gestalt annimmt.
- Kommunikation mit den Kunden: Anwendungsfalldiagramme sind wegen ihrer einfachen Notation ideal für die Kommunikation zwischen Entwickler und Kunde geeignet.
- Erstellung von Testfällen: Aus den Szenarios eines Anwendungsfalls können die entsprechenden Testfälle abgeleitet werden.

Definition

Ein Anwendungsfalldiagramm beschreibt die Funktion eines Systems von einem externen Beobachter aus gesehen. Der Schwerpunkt liegt darauf, was das System tut, und nicht darauf, wie es dies macht.

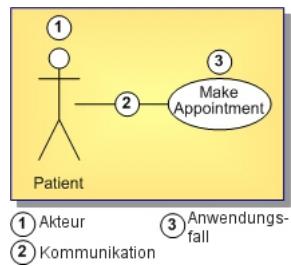
Das Anwendungsfalldiagramm ist eng an ein bestimmtes Szenario angelehnt. Ein Szenario ist ein Beispiel dafür, was geschieht, wenn jemand mit dem System interagiert.

Beispieldiagramm

Das folgende Szenario beschreibt einen Vorgang in einer Klinik:

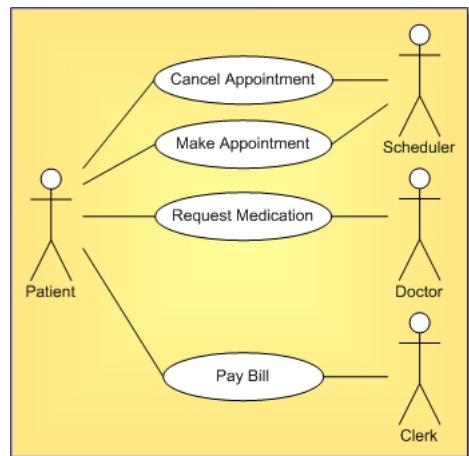
Ein Patient ruft in der Klinik an, um einen Termin für seine jährliche Untersuchung zu vereinbaren. Der Mitarbeiter am Empfang sucht im Terminkalender nach dem nächsten freien Termin und trägt den Patienten ein.

Ein Anwendungsfall ist eine Zusammenfassung der Szenarios für eine bestimmte Aufgabe oder ein bestimmtes Ziel. Ein Akteur ist die Person (bzw. das Objekt), von der die Aktivitäten in dieser Aufgabe eingeleitet werden. Akteure sind einfach gesagt Rollen, die von Personen oder Objekten übernommen werden. Das folgende Diagramm zeigt den Anwendungsfall Make Appointment (Termin vereinbaren) für die Klinik. Der Akteur ist ein Patient. Die Verbindung zwischen Akteur und Anwendungsfall ist eine Kommunikationsbeziehung (Kurzform Kommunikation).



Die Akteure werden als Strichmännchen angezeigt. Die Anwendungsfälle sind die Ovale. Die Kommunikation wird durch Linien zwischen den Akteuren und den Anwendungsfällen dargestellt.

Ein Anwendungsfalldiagramm ist eine Ansammlung von Akteuren, Anwendungsfällen und ihren Kommunikationsbeziehungen. Das folgende Beispiel zeigt den Anwendungsfall "Termin vereinbaren" eines Diagramms mit vier Akteuren und vier Anwendungsfällen. Beachten Sie, dass ein Anwendungsfall mehrere Akteure haben kann.



4.2.6.2.2 Elemente in einem UML 1.5-Anwendungsfalldiagramm

In der folgenden Tabelle sind die Elemente eines UML 1.5-Anwendungsfalldiagramms aufgeführt, die in der der Tool-Palette zur Verfügung stehen.

Elemente in einem UML 1.5-Anwendungsfalldiagramm

Name	Typ
Akteur	Knoten
Anwendungsfall	Knoten
Kommunikation	Beziehung
Extension	Beziehung
Inklusion	Beziehung
Generalisierung	Beziehung

Systemgrenze	Knoten
Knoten nach Pattern	Öffnet den Pattern-Experten (siehe Seite 1277).
Beziehung nach Pattern	Öffnet den Pattern-Experten (siehe Seite 1277).
Hinweis	Annotation
Hinweisbeziehung	Annotationsbeziehung

4.2.6.2.3 Erweiterungspunkt

Ein **Erweiterungspunkt** verweist auf die Position in einem Anwendungsfall, an der Aktionsfolgen aus einem anderen Anwendungsfall eingefügt werden können.

Erweiterungspunkte bestehen aus einem eindeutigen Namen in einem Anwendungsfall und einer Beschreibung der Position innerhalb des Verhaltens des Anwendungsfalls.

In einem Anwendungsfalldiagramm werden die Erweiterungspunkte mit der Überschrift **Erweiterungspunkte** angezeigt (in Fettschrift in der Diagrammansicht).

Siehe auch

UML 1.5-Anwendungsfalldiagramm (siehe Seite 1300)

UML 2.0-Anwendungsfalldiagramm (siehe Seite 1314)

4.2.6.3 UML 1.5-Interaktionsdiagramm

Dieser Abschnitt beschreibt die Elemente eines UML 1.5-Sequenz- und Kollaborationsdiagramms.

4.2.6.3.1 Definition eines UML 1.5-Sequenzdiagramms

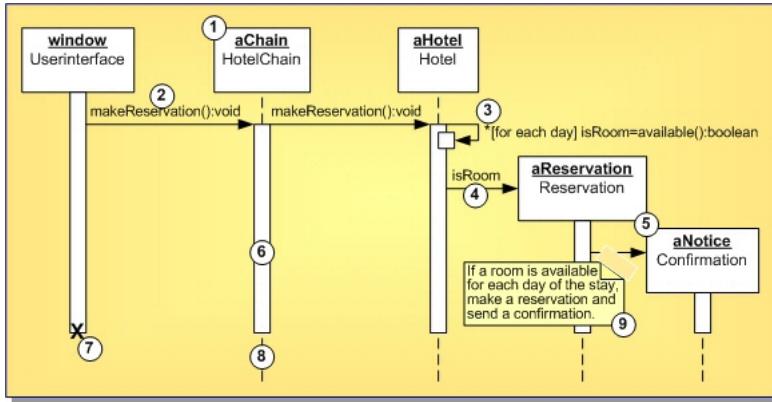
Klassendiagramme sind statische Modellansichten. Interaktionsdiagramme sind hingegen dynamisch und beschreiben die Aktionen der Objekte.

Definition

Ein Sequenzdiagramm ist ein Interaktionsdiagramm, das genau zeigt, wie die Operationen ausgeführt werden: welche Nachrichten wann gesendet werden. Diese Diagramme sind zeitlich organisiert. Der Zeitablauf erfolgt im Diagramm von oben nach unten. Die an der Operation beteiligten Objekte werden von links nach rechts entsprechend dem Zeitpunkt, an dem Sie am Nachrichtenaustausch teilnehmen, angezeigt.

Beispieldiagramm

Das folgende Sequenzdiagramm zeigt ein Hotelreservierungssystem. Der Nachrichtenaustausch wird durch ein Reservierungsfenster (UserInterface) eingeleitet.



- ① Objekt ③ Iteration ⑤ Erstellung ⑦ Löschung ⑨ Hinweis
 ② Nachricht ④ Bedingung ⑥ Aktivierungsleiste ⑧ Lebenslinie

Das UserInterface-Objekt sendet die Nachricht `makeReservation()` an ein HotelChain-Objekt. Das HotelChain-Objekt sendet die Nachricht `makeReservation()` an ein Hotel-Objekt. Wenn Zimmer frei sind, wird eine Reservation (Reservierung) und eine Confirmation (Bestätigung) vorgenommen.

Die vertikalen gestrichelten Linien sind Lebenslinien, die die Lebenszeit des jeweiligen Objekts anzeigen. Die Pfeile sind ausgehende Nachrichten. Sie führen vom sendenden Objekt zum oberen Rand der Aktivierungsleiste der Nachricht auf der Lebenslinie des Empfängers. Die Aktivierungsleiste stellt die Ausführungsdauer der Nachricht dar.

In diesem Beispieldiagramm sendet das Hotel-Objekt eine Nachricht an sich selbst, um festzustellen, ob ein Zimmer frei ist. Ist dies der Fall, erstellt das Hotel-Objekt die Objekte Reservation und Confirmation. Das Sternchen an der Nachricht weist auf eine Iteration hin (um sicherzustellen, dass an jedem Tag des Hotelaufenthalts ein Zimmer frei ist). Der Ausdruck in eckigen Klammern ([]) ist eine Bedingung.

Das Rechteck mit dem Eselsohr ist ein Hinweiselement mit erklärendem Text. Diese Elemente können in alle Arten von UML-Diagrammen eingefügt werden.

4.2.6.3.2 Definition eines UML 1.5-Kollaborationsdiagramms

Klassendiagramme sind statische Modellansichten. Interaktionsdiagramme sind hingegen dynamisch und beschreiben die Aktionen der Objekte.

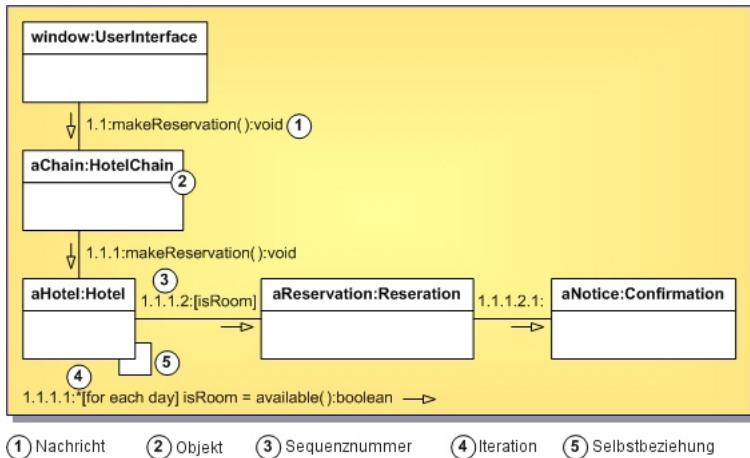
Definition

Kollaborationsdiagramme sind wie Sequenzdiagramme ebenfalls Interaktionsdiagramme. Sie zeigen jedoch die Objektrollen und nicht den zeitlichen Ablauf des Nachrichtenaustauschs.

In einem Sequenzdiagramm werden die Objektrollen oben in den Elementen und die Nachrichten als Beziehungslinien angezeigt. In einem Kollaborationsdiagramm (siehe unten) werden die Rechtecke der Objektrollen entweder mit dem Klassen- oder Objektnamen (oder beiden) bezeichnet. Vor den Klassennamen befindet sich ein Doppelpunkt (:).

Beispieldiagramm

Jede Nachricht in einem Kollaborationsdiagramm hat eine Sequenznummer. Die Nachricht der obersten Ebene hat die Nummer 1. Alle Nachrichten derselben Ebene (die im selben Aufruf gesendet werden) erhalten die gleiche Hauptnummer (1) und werden dann nach dem Punkt entsprechend ihrem Auftreten durchnummieriert (1, 2 usw.).



4.2.6.3.3 Elemente in einem UML 1.5-Interaktionsdiagramm

In der folgenden Tabelle sind die Elemente eines UML 1.5-Interaktionsdiagramms aufgeführt, die in der Tool Palette zur Verfügung stehen.

Elemente in einem UML 1.5-Interaktionsdiagramm

Name	Typ
Objekt	Knoten
Akteur	Knoten
Nachricht	Beziehung
Selbstbenachrichtigung	Beziehung
Nachricht mit Sendezeit	Beziehung, nur Sequenz
Konditionaler Block	Knoten, nur Sequenz
Rückgabe	Beziehung, nur Sequenz
Assoziation	Beziehung, nur Kollaboration
Aggregation	Beziehung, nur Kollaboration
Knoten nach Pattern	Öffnet den Pattern-Experten (siehe Seite 1277).
Beziehung nach Pattern	Öffnet den Pattern-Experten (siehe Seite 1277).
Hinweis	Annotation
Hinweisbeziehung	Annotationsbeziehung

Mit dem Befehl **Hinzufügen > Verknüpfung** können Sie einem Interaktionsdiagramm auch Verknüpfungen hinzufügen. Es sind jedoch keine Verknüpfungen mit Elementen in den anderen Interaktionsdiagrammen zulässig.

4.2.6.3.4 Konditionaler Block

Die konditionale Blockanweisung ist eine flexible Möglichkeit, ein Sequenzdiagramm zu erweitern. Folgende Anweisungen werden unterstützt:

- if
- else

- for
- foreach
- while
- do while
- try
- catch
- finally
- switch
- case
- default

4.2.6.3.5 UML 1.5-Nachricht

Die Nachrichtenbeziehungen in einem Sequenzdiagramm sind standardmäßig von oben nach unten durchnummieriert. Sie können die Nachrichten auch anders anordnen.

Eine "Selbstbenachrichtigung" ist eine Nachricht von einem Objekt an sich selbst:

Siehe auch

Nachrichten in UML 2.0 ([siehe Seite 1318](#))

4.2.6.3.6 Aktivierungsleiste

Together zeigt automatisch die Aktivitätsdauer von Nachrichten im Diagramm an. Wenn Sie eine Nachrichtenbeziehung zum Zielobjekt herstellen, wird die Aktivierungsleiste automatisch erstellt.

Sie können die Aktivitätsdauer einer Nachricht verlängern oder verkürzen, indem Sie die obere bzw. untere Linie der Leiste nach oben oder unten ziehen. Je länger die Leiste ist, desto länger ist die betreffende Nachricht aktiv.

4.2.6.3.7 Verschachtelte Nachricht

Sie können Nachrichten verschachteln, indem Sie eine Aktivierungsleiste als Ursprung der Nachrichtenbeziehungen verwenden. Bei den verschachtelten Nachrichten wird die Nummerierung der übergeordneten Nachrichten weitergeführt.

Wenn beispielsweise die übergeordnete Nachricht die Nummer 1 hat, erhält ihre erste verschachtelte Nachricht die Nummer 1.1. Sie können auch Nachrichtenbeziehungen zurück zu den übergeordneten Aktivierungsleisten erstellen.

Siehe auch

UML 1.5-Nachricht ([siehe Seite 1305](#))

4.2.6.4 UML 1.5-Zustandsdiagramm

Dieser Abschnitt beschreibt die Elemente eines UML 1.5-Zustandsdiagramms.

4.2.6.4.1 Definition eines UML 1.5-Zustandsdiagramms

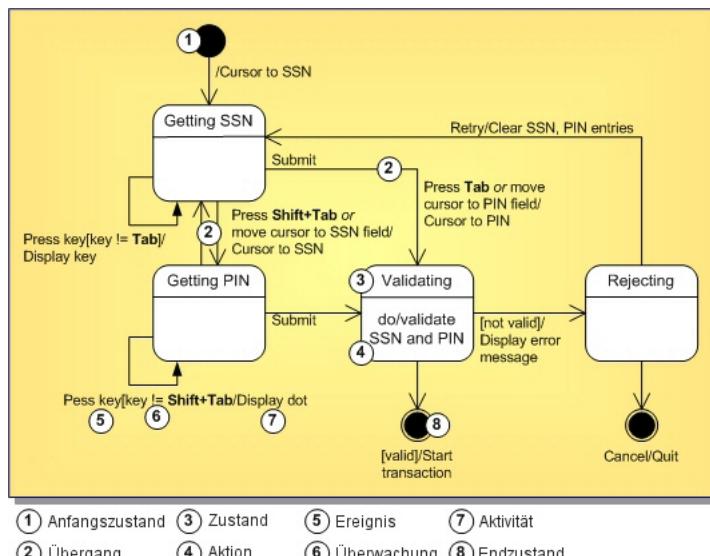
Dieses Thema beschreibt das UML 1.5 Zustandsdiagramm.

Definition

Objekte verfügen über ein Verhalten und über Zustände. Der Zustand eines Objekts ist von seiner aktuellen Aktivität oder Bedingung abhängig. In einem Zustandsdiagramm werden die möglichen Zustände des Objekts und die Übergänge zu einem anderen Zustand dargestellt.

Beispieldiagramm

Das folgende Diagramm modelliert die Anmeldung bei einem Online-Banksystem. Der Benutzer muss eine gültige Sozialversicherungsnummer sowie eine PIN-Nummer eingeben und dann die Daten zur Überprüfung senden. Die Anmeldung kann mit vier Zuständen beschrieben werden: SVN abrufen, PIN abrufen, Überprüfen und Zurückweisen. Jeder Zustand enthält vollständige Übergangsdefinitionen, die den nachfolgenden Zustand bestimmen.



- ① Anfangszustand ③ Zustand ⑤ Ereignis ⑦ Aktivität
 ② Übergang ④ Aktion ⑥ Überwachung ⑧ Endzustand

Zustände werden als abgerundete Rechtecke angezeigt. Die Übergänge werden durch Pfeile zwischen den Zuständen dargestellt. Ereignisse oder Bedingungen, die zu einem Übergang führen, werden neben den Pfeillinien angezeigt. Im Diagramm sind zwei Selbstübergänge vorhanden: SVN abrufen und PIN abrufen. Der Anfangszustand (schwarzer Kreis) ist ein reines Anzeigeelement zum Starten der Aktion. Die Endzustände dienen ebenfalls nur der Anzeige und beenden die Aktion.

Die aufgrund eines Ereignisses oder einer Bedingung ausgeführte Aktion wird im Format /action angegeben. Während sich das Objekt im Zustand Überprüfen befindet, wartet es nicht darauf, dass ein externes Ereignis einen Übergang auslöst. Es führt stattdessen eine Aktivität durch. Das Ergebnis dieser Aktivität bestimmt dann den nächsten Zustand.

4.2.6.4.2 Elemente in einem UML 1.5-Zustandsdiagramm

In der folgenden Tabelle sind die Elemente eines UML 1.5-Zustandsdiagramms aufgeführt, die in der der Tool Palette zur Verfügung stehen.

Elemente in einem UML 1.5-Zustandsdiagramm

Name	Typ
Status	Knoten
Anfangszustand	Knoten
Endzustand	Knoten
Historie	Knoten
Objekt	Knoten

Übergang	Beziehung
Horizontale Aufspaltung/Zusammenführung	Knoten
Vertikale Aufspaltung/Zusammenführung	Knoten
Knoten nach Pattern	Öffnet den Pattern-Experten (siehe Seite 1277).
Beziehung nach Pattern	Öffnet den Pattern-Experten (siehe Seite 1277).
Hinweis	Annotation
Hinweisbeziehung	Annotationsbeziehung

4.2.6.4.3 Zustand

Ein Zustand stellt eine Situation dar, in der eine (normalerweise implizite) invariante Bedingung gegeben ist. Die Invariante kann eine statische Situation sein, z.B. das Warten eines Objekts auf ein externes Ereignis, oder eine dynamische Bedingung repräsentieren, etwa die Ausführung einer Aktivität. Ein zu konstruierendes Modellelement kann beispielsweise beim Start der Aktivität in den Zustand eintreten und nach ihrem Abschluss den Zustand wieder verlassen.

Aktionen

Die Ein- und Austrittsaktionen werden beim Eintritt in einen bzw. Austritt aus einem Zustand ausgeführt.

Sie können diese Aktionen in Zustandsdiagrammen als Knoten oder stereotypisierte interne Übergänge erstellen.

Zusammengesetzte (verschachtelte) Zustände

Sie können zusammengesetzte Zustände erstellen, indem Sie in einem Zustand eine oder mehrere Zustandsebenen verschachteln. Sie können in einen Zustand auch Anfangs-/Endzustände oder einen Historienzustand einfügen und Übergänge zwischen den Unterzuständen erstellen.

Siehe auch

UML 1.5-Zustandsdiagramm – Referenz (siehe Seite 1305)

UML 1.5-Aktivitätsdiagramm – Referenz (siehe Seite 1308)

UML 2.0-Zustandsmaschinendiagramm – Referenz (siehe Seite 1321)

4.2.6.4.4 Übergang

Jeder Zustand und jede Aktivität wird durch einen einzigen Übergang mit dem nächsten Zustand bzw. der nächsten Aktivität verbunden.

Ein Übergang bringt die Operation von einem Zustand in einen anderen und stellt die Reaktion auf ein bestimmtes Ereignis dar. Sie können Zustände mit Übergängen verbinden und in Zuständen interne Übergänge erstellen.

Interner Übergang

Ein interner Übergang ist eine Möglichkeit zur Behandlung von Ereignissen, ohne einen Zustand (oder eine Aktivität) zu verlassen und seine Aus- oder Eintrittsaktionen weiterzugeben. Interne Übergänge können einem Zustands- oder Aktivitätselement hinzugefügt werden.

Ein interner Übergang ist eine Möglichkeit zur Behandlung von Ereignissen, ohne einen Zustand zu verlassen und seine Aus- oder Eintrittsaktionen weiterzugeben.

Selbstübergang

Bei einem Selbstübergang verlässt das Objekt den aktuellen Zustand (bzw. Aktivität), gibt die Austrittsaktionen(en) weiter, tritt dann erneut in den Zustand ein und leitet zuletzt die Eintrittsaktion(en) weiter. Selbstübergänge können Aktivitäts- und

Zustandselementen in einem Aktivitätsdiagramm hinzugefügt werden.

Mehrfachübergang

Ein Übergang kann sich in zwei oder mehr Übergänge verzweigen, die sich gegenseitig ausschließen.

Ein Übergang kann sich in zwei oder mehr parallele Aktivitäten aufspalten. Die Aufspaltung und die nachfolgende Zusammenführung der aufgespalteten Threads wird durch ausgefüllte Balken angezeigt.

Ein Übergang kann mehrere Quellzustände (d.h. er ist eine Zusammenführung mehrerer gleichzeitiger Zustände) oder mehrere Zielzustände haben (d.h. er ist eine Aufspaltung in mehrere gleichzeitige Zustände).

Sie können Mehrfachübergänge in Ihren Zustands- und Aktivitätsdiagrammen vertikal oder horizontal anzeigen. In der Symbolleiste beider Diagrammarten sind separate Schaltflächen für vertikale und horizontale Aufspaltungen/Zusammenführungen vorhanden. Die Semantik ist bei beiden Ausrichtungen identisch.

Bedingungsausdrücke

Alle Übergänge (auch die internen) verfügen über Bedingungen (logische Ausdrücke), die definieren, ob der Übergang erfolgt. Sie können einem Übergang auch einen Effekt zuweisen, eine optionale Aktivität, die beim Auslösen des Übergangs ausgeführt wird. Die Bedingungen werden in eckigen Klammern (z.B. "[false]") in der Nähe der Übergangsbeziehung angezeigt. Die Effektaktivität wird neben der Bedingung angezeigt. Bedingung und Effekt können im Objektinspektor festgelegt werden.

Die von einer Verzweigung ausgehenden Übergänge sind mit Bedingungsausdrücken (in []) versehen. Eine Verzweigung und die nachfolgende Zusammenfassung, die das Ende der Verzweigung kennzeichnet, werden durch leere Rauten dargestellt.

Siehe auch

UML 1.5-Zustandsdiagramm ([siehe Seite 1305](#))

UML 1.5-Aktivitätsdiagramm ([siehe Seite 1308](#))

UML 2.0-Zustandsmaschinendiagramm ([siehe Seite 1321](#))

4.2.6.4.5 Verzögertes Ereignis

Ein **verzögertes Ereignis** ist wie ein interner Übergang, der das Ereignis bearbeitet und es in eine interne Warteschlange stellt, bis es verwendet oder verworfen wird.

Stellen Sie sich ein verzögertes Ereignis so vor, dass ein interner Übergang das Ereignis bearbeitet und es in eine interne Warteschlange stellt, bis es verwendet oder verworfen wird. Verzögerte Ereignisse können einem Zustands- oder Aktivitätselement hinzugefügt werden.

Siehe auch

UML 1.5-Zustandsdiagramm ([siehe Seite 1305](#))

UML 1.5-Aktivitätsdiagramm ([siehe Seite 1308](#))

UML 2.0-Zustandsmaschinendiagramm ([siehe Seite 1321](#))

4.2.6.5 UML 1.5-Aktivitätsdiagramm

Dieser Abschnitt beschreibt die Elemente eines UML 1.5-Aktivitätsdiagramms.

4.2.6.5.1 Elemente in einem UML 1.5-Aktivitätsdiagramm

In der folgenden Tabelle sind die Elemente eines UML 1.5-Aktivitätsdiagramms aufgeführt, die in der der Tool Palette zur

Verfügung stehen.

Elemente in einem UML 1.5-Aktivitätsdiagramm

Name	Typ
Aktivität	Knoten
Entscheidung	Knoten
Signalempfang	Knoten
Signalsendung	Knoten
Status	Knoten
Anfangszustand	Knoten
Endzustand	Knoten
Historie	Knoten
Objekt	Knoten
Übergang	link
Horizontale Aufspaltung/Zusammenführung	Knoten
Vertikale Aufspaltung/Zusammenführung	Knoten
Verantwortlichkeitsbereich	Knoten
Objektfluss	link
Knoten nach Pattern	Öffnet den Pattern-Experten (siehe Seite 1277).
Beziehung nach Pattern	Öffnet den Pattern-Experten (siehe Seite 1277).
Hinweis	Annotation
Hinweisbeziehung	Annotationsbeziehung

4.2.6.5.2 Definition eines UML 1.5-Aktivitätsdiagramms

Dieses Thema beschreibt das UML 1.5 Aktivitätsdiagramm.

Definition

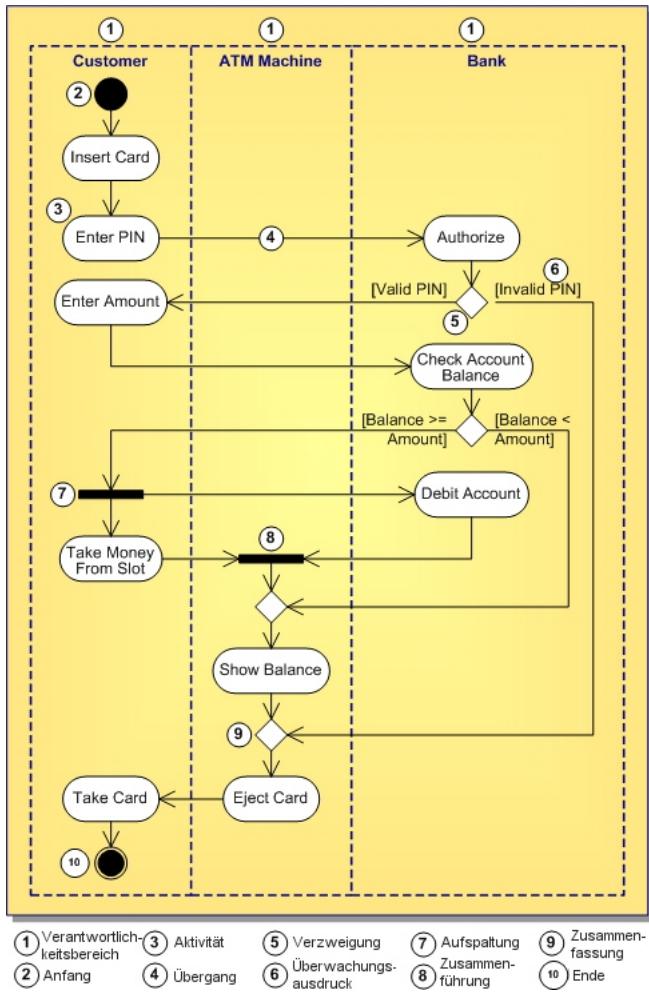
Ein Aktivitätsdiagramm ähnelt einem Flussdiagramm. Aktivitätsdiagramme und Zustandsdiagramme sind miteinander verwandt. Während aber ein Zustandsdiagramm das Verhalten eines Objekt in einem Vorgang beschreibt, modelliert ein Aktivitätsdiagramm den Ablauf der Aktivitäten des Vorgangs. Das Aktivitätsdiagramm zeigt, wie diese Aktivitäten voneinander abhängen.

Aktivitätsdiagramme können in Verantwortlichkeitsbereiche unterteilt werden, die bestimmen, welches Objekt für eine Aktivität zuständig ist.

Beispieldiagramm

Das unten gezeigte Aktivitätsdiagramm beschreibt den folgenden Vorgang: "Geld über einen Geldautomaten von einem Bankkonto abheben." Die drei beteiligten Klassen der Aktivität sind Kunde, Geldautomat und Bank. Der Vorgang beginnt mit dem schwarzen Anfangskreis oben und endet mit dem konzentrischen weißen/schwarzen Kreis unten. Die Aktivitäten werden durch abgerundete Rechtecke angezeigt.

Die drei beteiligten Klassen (Personen usw.) der Aktivität sind Kunde, Geldautomat und Bank. Der Vorgang beginnt mit dem schwarzen Anfangskreis oben und endet mit dem konzentrischen weißen/schwarzen Kreis unten. Die Aktivitäten werden durch abgerundete Rechtecke angezeigt.



4.2.6.6 UML 1.5-Komponentendiagramm

Dieser Abschnitt beschreibt die Elemente eines UML 1.5-Komponentendiagramms.

4.2.6.6.1 Definition eines UML 1.5-Komponentendiagramms

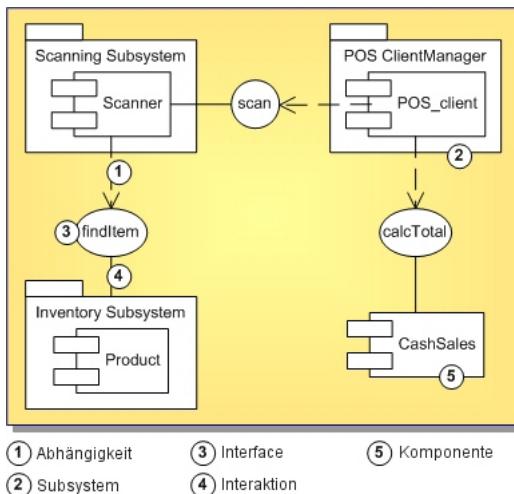
Mit Hilfe von Komponenten- und Verteilungsdiagrammen kann die physische Architektur eines computergestützten Systems beschrieben werden. Komponentendiagramme zeigen die Abhängigkeiten und Interaktionen zwischen den Softwarekomponenten.

Definition

Eine Komponente ist ein Container für logische Elemente und repräsentiert die ausführenden Elemente eines Systems. Komponenten können auch über ihre Schnittstellen auf die Dienste anderer Komponenten zugreifen. Mit Hilfe von Komponenten werden in der Regel die logischen Quelltextpakete (Arbeitskomponenten), der Binärkode (Verteilungskomponenten) oder die ausführbaren Dateien (Ausführungskomponenten) grafisch dargestellt.

Beispieldiagramm

Das folgende Komponentendiagramm zeigt die Abhangigkeiten und Interaktionen zwischen den Softwarekomponenten eines Registrierkassensystems.



4.2.6.6.2 Elemente in einem UML 1.5-Komponentendiagramm

In der folgenden Tabelle sind die Elemente eines UML 1.5-Komponentendiagramms aufgeführt, die in der der Tool Palette zur Verfügung stehen.

Elemente in einem UML 1.5-Komponentendiagramm

Name	Typ
Subsystem	Knoten
Komponente	Knoten
Interface	Knoten
Support	Beziehung
Abhängigkeit	Beziehung
Knoten nach Pattern	Öffnet den Pattern-Experten (siehe Seite 1277).
Beziehung nach Pattern	Öffnet den Pattern-Experten (siehe Seite 1277).
Hinweis	Annotation
Hinweisbeziehung	Annotationsbeziehung

4.2.6.7 UML 1.5-Verteilungsdiagramm

Dieser Abschnitt beschreibt die Elemente eines UML 1.5-Verteilungsdiagramms.

4.2.6.7.1 Definition eines UML 1.5-Verteilungsdiagramms

Mit Hilfe von Komponenten- und Verteilungsdiagrammen kann die physische Architektur eines computergestützten Systems beschrieben werden.

Verteilungsdiagramme bestehen aus Knoten, die durch Kommunikationspfade verbunden werden, um die Konfiguration der Software und Hardware darzustellen.

Komponenten sind u. a. folgende physische Softwareeinheiten:

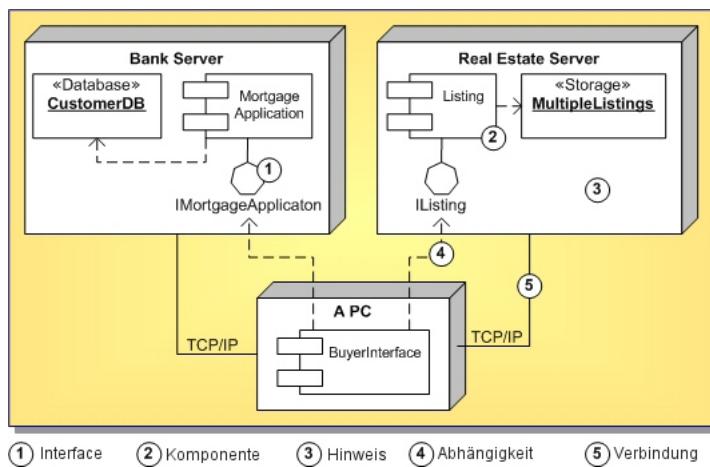
- Externe Bibliotheken
- Betriebssysteme
- Virtuelle Maschinen

Definition

Die physische Hardware wird durch Knoten dargestellt. Jede Komponente gehört zu einem Knoten. Die Komponenten werden als Rechtecke angezeigt.

Beispieldiagramm

Das folgende Verteilungsdiagramm zeigt die Beziehungen der Hardware- und Softwarekomponenten bei einer Immobilientransaktion.



4.2.6.7.2 Elemente in einem UML 1.5-Verteilungsdiagramm

In der folgenden Tabelle sind die Elemente eines UML 1.5-Verteilungsdiagramms aufgeführt, die in der der Tool Palette zur Verfügung stehen.

Elemente in einem UML 1.5-Verteilungsdiagramm

Name	Typ
Knoten	Knoten
Komponente	Knoten
Interface	Knoten
Support	Beziehung
Aggregation	Beziehung
Objekt	Knoten
Assoziation	Beziehung
Abhängigkeit	Beziehung
Knoten nach Pattern	Öffnet den Pattern-Experten (siehe Seite 1277).
Beziehung nach Pattern	Öffnet den Pattern-Experten (siehe Seite 1277).
Hinweis	Annotation
Hinweisbeziehung	Annotationsbeziehung

4.2.7 UML 2.0-Referenz

Dieser Abschnitt enthält Referenzmaterial für UML 2.0-Diagramme.

4.2.7.1 UML 2.0-Klassendiagramme

Dieser Abschnitt beschreibt die Elemente eines UML 2.0-Klassendiagramms.

4.2.7.1.1 Definition eines UML 2.0-Klassendiagramms

UML 2.0-Klassendiagramme bieten dieselben Möglichkeiten wie die UML 1.5-Diagramme.

Die UML 2.0-Klassendiagramme verfügen über neue Diagrammelemente wie Ports, Slots sowie bereitgestellte und erforderliche Interfaces.

Gemäß dem UML 2.0-Standard kann eine Instanzspezifikation einen oder mehrere Klassifizierer instantiiieren. Als Klassifizierer können Klassen, Interfaces oder Komponenten verwendet werden.

Interfaces

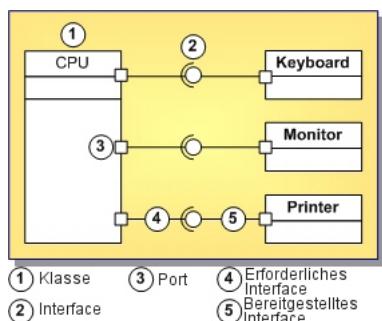
Eine Klasse implementiert ein Interface wie in UML 1.5-Klassendiagrammen über eine Generalisierungs-/Implementierungsbeziehung. Außerdem gibt es bereitgestellte und erforderliche Interfaces. Interfaces können in Klassendiagrammen als Rechtecke oder Kreise angezeigt werden. Um Ihre Diagramme übersichtlicher zu machen, können Sie Interfaces anzeigen oder verbergen.

Tip: Wenn Sie über ein bereitgestelltes Interface eine Beziehung zwischen einem Klassen- und einem Interface-Element herstellen, wird eine normale Generalisierungs-/Implementierungsbeziehung erstellt. Um ein bereitgestelltes Interface zu erstellen, stellen Sie die Verbindung mit einem Port der Client-Klasse her.

UML 2.0-Klassendiagramme unterstützen die Kugelgelenksnotation für die bereitgestellten und erforderlichen Interfaces. Wenn Sie im Kontextmenü des Interface auf den Befehl Als Kreis anzeigen klicken, wird zwischen Client-Klasse und Anbieter-Interface ein "Lollipop" angezeigt.

Beispieldiagramm

Die folgende Abbildung zeigt ein Klassendiagramm mit den neuen Elementen.



4.2.7.1.2 Elemente in einem UML 2.0-Klassendiagramm

In der folgenden Tabelle sind die Elemente eines UML 2.0-Klassendiagramms aufgeführt, die in der Tool Palette zur Verfügung stehen.

Elemente in einem UML 2.0-Klassendiagramm

Name	Typ
Paket	Knoten
Klasse	Knoten
Interface	Knoten
Assoziationsklasse	Knoten
Port	Knoten
Instanzspezifikation	Knoten
Generalisierung/Implementierung	Beziehung
Bereitgestelltes Interface	Beziehung
Erforderliches Interface	Beziehung
Assoziation	Beziehung
Assoziationsende	Beziehung
Abhangigkeit	Beziehung
Knoten nach Pattern	Offnet den Pattern-Experten (siehe Seite 1277).
Beziehung nach Pattern	Offnet den Pattern-Experten (siehe Seite 1277).
Einschrenkung	OCL-Knoten
Einschrenkungsbeziehung	OCL-Beziehung
Hinweis	Annotation
Hinweisbeziehung	Annotationsbeziehung

4.2.7.2 UML 2.0-Anwendungsfalldiagramm

Dieser Abschnitt beschreibt die Elemente eines UML 2.0-Anwendungsfalldiagramms.

4.2.7.2.1 Definition des UML 2.0-Anwendungsfalldiagramms

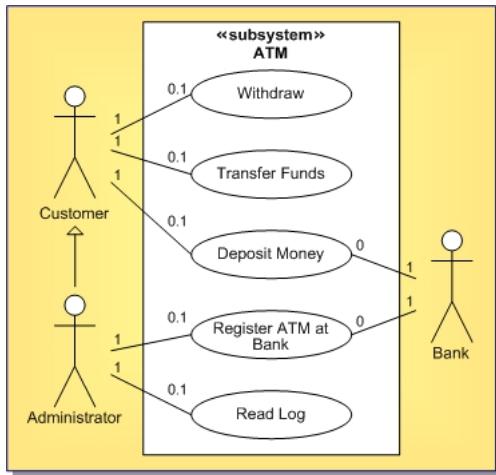
Das Diagramm wurde folgendem Buch entnommen: *Unified Modeling Language: Superstructure Version 2.0. vom August 2003 (Seite 536).*

Definition

Mit einem Anwendungsfalldiagramm konnen Sie ein System und dessen Interaktionen beschreiben. Die wichtigsten Elemente dazu sind Akteure, Anwendungsfalle und Subjekte. Ein Subjekt repräsentiert ein System, mit dem die Akteure und anderen Subjekte interagieren. Das erforderliche Verhalten des Subjekts wird durch die Anwendungsfalle beschrieben.

Beispieldiagramm

Das folgende Diagramm enthalt die Akteure und Anwendungsfalle fur einen Geldautomaten.



4.2.7.2.2 Elemente in einem UML 2.0-Anwendungsfalldiagramm

In der folgenden Tabelle sind die Elemente eines UML 2.0-Anwendungsfalldiagramms aufgeführt, die in der der Tool Palette zur Verfügung stehen.

Elemente in einem UML 2.0-Anwendungsfalldiagramm

Name	Typ	Beschreibung
Akteur	Knoten	
Subjekt	Knoten	
Anwendungsfall	Knoten	
Erweiterungspunkt	Knoten	Erstellen Sie Erweiterungspunkte in einem Anwendungsfall, um sein Verhalten durch das Verhalten eines anderen Anwendungsfalls zu erweitern. Dieses Element steht im Kontextmenü eines Anwendungsfalls zur Verfügung.
Extends	Beziehung	
Inklusion	Beziehung	
Generalisierung	Beziehung	
Assoziation	Beziehung	
Knoten nach Pattern	Öffnet den Pattern-Experten (siehe Seite 1277).	
Beziehung nach Pattern	Öffnet den Pattern-Experten (siehe Seite 1277).	
Hinweis	Annotation	
Hinweisbeziehung	Annotationsbeziehung	

4.2.7.3 UML 2.0-Interaktionsdiagramm

Dieser Abschnitt beschreibt die Elemente eines UML 2.0-Kommunikations- und Sequenzdiagramms.

4.2.7.3.1 Definition eines UML 2.0-Sequenzdiagramms

Interaktionen können in Together-Projekten mithilfe der zwei gebräuchlichsten Interaktionsdiagramme grafisch dargestellt werden: Sequenz- und Kommunikationsdiagramm.

Die gebräuchlichste Art eines Interaktionsdiagramms ist das Sequenzdiagramm, das den Nachrichtenverkehr zwischen mehreren Lebenslinien darstellt. Ein Sequenzdiagramm beschreibt eine Interaktion, indem es die Sequenz der ausgetauschten Nachrichten darstellt.

Siehe auch

Referenz eines UML 2.0-Interaktionsdiagramms ([siehe Seite 1315](#))

Interaktion ([siehe Seite 1317](#))

4.2.7.3.2 Definition eines UML 2.0-Kommunikationsdiagramms

Interaktionen können in Together-Projekten mithilfe der zwei gebräuchlichsten Interaktionsdiagramme grafisch dargestellt werden: Sequenz- und Kommunikationsdiagramm.

Kommunikationsdiagramme stellen die Interaktion zwischen Lebenslinien dar, wobei besonders die Architektur der internen Struktur und ihre Korrespondenz mit der durchgehenden Nachricht von Bedeutung ist. Die Sequenz der Nachrichten ergibt sich aus einem Sequenz-Nummerierungsschema.

Siehe auch

Referenz eines UML 2.0-Interaktionsdiagramms ([siehe Seite 1315](#))

Interaktion ([siehe Seite 1317](#))

4.2.7.3.3 Elemente in einem UML 2.0-Sequenzdiagramm

In der folgenden Tabelle sind die Elemente eines UML 2.0-Sequenzdiagramms aufgeführt, die in der Tool Palette zur Verfügung stehen.

Elemente in einem UML 2.0-Sequenzdiagramm

Name	Typ
Lebenslinie	Knoten
Ausführungsspezifikation	Knoten
Kombiniertes Fragment	Knoten
Zustandsinvariante	Knoten
Nachricht	Beziehung
Interaktionsverwendung	Knoten
Knoten nach Pattern	Öffnet den Pattern-Experten (siehe Seite 1277).
Beziehung nach Pattern	Öffnet den Pattern-Experten (siehe Seite 1277).
Hinweis	Annotation
Hinweisbeziehung	Annotationsbeziehung

Sequenzdiagramme können zwar Verknüpfungen mit Elementen in anderen Diagrammen enthalten, jedoch werden keine Verknüpfungen mit Elementen in anderen Interaktionsdiagrammen unterstützt.

Bei Interaktionsdiagrammen werden in der Modellansicht Hilfselemente angezeigt, die in der Diagrammansicht nicht zu sehen sind. Diese Elemente werden lediglich zur Darstellung der Diagrammstruktur verwendet. Sie können zwar bearbeitet werden, es wird aber ausdrücklich davon abgeraten, dies zu tun. Die Integrität der Interaktionsdiagramme ist sonst nicht gewährleistet.

4.2.7.3.4 Elemente in einem UML 2.0-Kommunikationsdiagramm

In der folgenden Tabelle sind die Elemente eines UML 2.0-Kommunikationsdiagramms aufgeführt, die in der Tool Palette zur Verfügung stehen.

Elemente in einem UML 2.0-Kommunikationsdiagramm

Name	Typ
Lebenslinie	Knoten
Nachricht	Beziehung
Knoten nach Pattern	Öffnet den Pattern-Experten (siehe Seite 1277).
Beziehung nach Pattern	Öffnet den Pattern-Experten (siehe Seite 1277).
Hinweis	Annotation
Hinweisbeziehung	Annotationsbeziehung

Bei Interaktionsdiagrammen werden in der Modellansicht Hilfselemente angezeigt, die in der Diagrammansicht nicht zu sehen sind. Diese Elemente werden lediglich zur Darstellung der Diagrammstruktur verwendet. Sie können zwar bearbeitet werden, es wird aber ausdrücklich davon abgeraten, dies zu tun. Die Integrität der Interaktionsdiagramme ist sonst nicht gewährleistet.

4.2.7.3.5 Interaktion

Mit Together können Sie Interaktionen zur genauen Beschreibung und Analyse der prozessübergreifenden Kommunikation erstellen.

Interaktionen können in Together-Projekten mithilfe der zwei gebräuchlichsten Interaktionsdiagramme grafisch dargestellt werden: Sequenz- und Kommunikationsdiagramm. Andererseits können Interaktionen auch ohne grafische Darstellung in Projekten vorhanden sein.

Interaktionsverwendung

Sie können in einer Interaktion auf andere Interaktionen in Ihrem Projekt verweisen. Die so genannten "Interaction use"-Elemente dienen genau diesem Zweck. Beachten Sie, dass die referenzierte Interaktion explizit aus dem Modell definiert oder einfach als Zeichenfolge eingegeben werden kann.

Jede Interaktionsverwendung ist mit einem schwarzen Punkt an ihrer Lebenslinie verankert. Bei diesem Punkt handelt es sich um ein separates Diagrammelement. Wenn eine Interaktionsverwendung auf mehrere Lebenslinien erweitert ist, können die Verankerungspunkte an allen Lebenslinien bis auf eine gelöscht werden. Interaktionsverwendungen müssen mit mindestens einer Lebenslinie verbunden sein.

Frame verankern

Sie können in Together kombinierte Fragmente und Interaktionen auf mehrere Lebenslinien erweitern. Dazu steht in der die Schaltfläche **Frame verankern** zur Verfügung.

Ein Frame kann an verschiedenen Stellen der Ziellebenslinie verankert werden. Sie brauchen nur bis zur gewünschten Position zu ziehen. Der Frame wird dann auf der Quelllebenslinie entsprechend verschoben.

Denken Sie aber daran, dass die Interaktion oder das kombinierte Fragmente nur an den Lebenslinien mit einem Punkt tatsächlich verankert ist. Die Lebenslinien, die lediglich überquert werden, sind nicht einbezogen. Die Verankerungspunkte sind

eigenständige Diagrammelemente, die ausgewählt und entfernt werden können.

Lebenslinie

Eine Lebenslinie definiert einen Teilnehmer der Interaktion. Sie wird in Sequenzdiagrammen als Rechteck mit einer darunter befindlichen gestrichelten Linie angezeigt.

Die Lebenslinien von Interaktionen können sich auf Elemente in der Klasse oder in Kompositionssstrukturdiagrammen beziehen. Wenn das betreffende Element mehrere Werte enthält, muss in der Lebenslinie angegeben werden, für welche Komponente des Elements sie verwendet wird.

Wenn die Lebenslinie zu einem verbindbaren Element gehört, auf eine andere Interaktion verweist oder der Typ ausgewählt wurde, steht in ihrem Kontextmenü das Untermenü **Auswählen** zur Verfügung. Über dieses Menü gelangen Sie zu dem Part, dem Typ oder der Dekomposition, die mit den Eigenschaften *represents*, *type* und *decomposition* ausgewählt wurden. Diese Eigenschaften werden im Objektinspektors festgelegt. Wenn die Eigenschaft *represents* zugewiesen wird, werden die Eigenschaften *type* und *part* deaktiviert.

Sie können diese Eigenschaften manuell zuweisen, indem Sie die Werte in die entsprechenden Felder des im Objektinspektors eingeben. Wenn die angegebenen Elemente nicht im Modell gefunden werden, wird ihr Name in einfachen Anführungszeichen angezeigt. Diese Referenzen beziehen sich nicht auf tatsächliche Elemente, und das Untermenü **Auswählen** steht für sie nicht zur Verfügung. Befinden sich die angegebenen Elemente im Modell, werden sie ohne Anführungszeichen angezeigt, und das Untermenü **Auswählen** steht zur Verfügung.

Zustandsinvariante

Eine Zustandsinvariante ist eine Einschränkung, die sich auf einer Lebenslinie befindet. Sie wird zur Laufzeit ausgewertet, bevor die nächste Ausführungsspezifikation ausgeführt wird. Zustandsinvarianten können auf zwei Arten zu Interaktionsdiagrammen hinzugefügt werden: als OCL-Ausdrücke oder als Referenzen auf die Zustandsdiagramme. Sie können mithilfe von Zustandsinvarianten Kommentare in Ihre Interaktionsdiagramme einfügen oder Interaktionen mit Zuständen verbinden.

Sie müssen dabei beachten, dass Together die Syntax der als OCL-Ausdrücke angegebenen Zustandsinvarianten prüft. Wenn die Syntax falsch ist oder es keinen gültigen Kontext gibt, wird die Einschränkung rot angezeigt. So müssen beispielsweise bei einer Lebenslinie die Eigenschaften *type* und *represents* zugewiesen sein, damit sie ein gültiger Kontext ist.

Siehe auch

Überblick über die OCL-Unterstützung (siehe Seite 1453)

UML 2.0-Interaktionsdiagramme (siehe Seite 1315)

4.2.7.3.6 UML 2.0-Nachricht

Aufrufnachrichten werden immer in Diagrammen angezeigt, Antwortnachrichten in der Regel nicht. Sie können die Antwortnachrichten jedoch ebenfalls anzeigen.

Nachrichten in unterschiedlichen Diagrammtypen

Nachrichten in Kommunikationsdiagrammen: Wenn Sie eine Nachricht zwischen zwei Lebenslinien erstellen, wird eine Beziehungslinie und darunter eine Liste der Nachrichten angezeigt. Die Beziehungslinie wird so lange angezeigt, wie mindestens eine Nachricht zwischen den Lebenslinien vorhanden ist.

Nachrichten in Sequenzdiagrammen: Nachrichten in Sequenzdiagrammen haben dieselben Eigenschaften wie die Nachrichten in Kommunikationsdiagrammen, mit ihnen können aber mehr Aktionen durchgeführt werden. Die nachfolgenden Informationen beziehen sich hauptsächlich auf Nachrichten in Sequenzdiagrammen.

Die Eigenschaften der Nachrichten in beiden Diagrammarten können im Objektinspektors bearbeitet werden.

Eigenschaften von Nachrichtenbeziehungen

Aufrufnachrichten verfügen über folgende Eigenschaften:

Eigenschaft	Beschreibung
Sequence number	Die Sequenznummer der Nachricht. Wenn Sie eine andere Nummer eingeben, wird die Aufrufnachricht entsprechend geändert.
Name	Der Name der Beziehung. Dieses Feld kann bearbeitet werden.
Qualified name	Der vollständige Name der Nachricht. Dieses Feld kann nicht geändert werden.
Stereotype	Hier legen Sie das Nachrichtenstereotyp fest. Der Stereotypname wird über der Beziehung angezeigt.
Signature	Hier geben Sie die Signatur (Name einer Operation oder eines Signals) der Nachricht an. Wenn Sie die Signatur einer Aufrufnachricht ändern, wird auch die Signatur der zugehörigen Antwortnachricht geändert.
Arguments	Die Argumente einer Operation, die der Nachricht zugeordnet ist. Dieses Feld kann bearbeitet werden.
Sort	Hier wählen Sie in einer Dropdown-Liste die Art der Synchronisierung aus. Mögliche Werte sind asynchCall, synchCall und asynchSignal. Die Nachrichtenbeziehung wird entsprechend aktualisiert. Asynchrone Benachrichtigungen unterliegen folgenden Einschränkungen: Sie können manchmal aufgrund von Frame-Einschränkungen nicht erstellt oder eingefügt werden. Ihre Ausführungsspezifikationen müssen sich immer auf einer Lebenslinie befinden.
Show reply message	Mit dieser Booleschen Option können Sie festlegen, ob ein gestrichelter Rückwärtspfeil angezeigt wird.
Commentary	Geben Sie hier einen Kommentar für die Nachrichtenbeziehung ein.

Antwortnachrichten verfügen über folgende Eigenschaften:

Eigenschaft	Beschreibung
Stereotype	Hier legen Sie das Nachrichtenstereotyp fest.
Attribute	Hier definieren Sie ein Attribut, dem der Rückgabewert der Nachricht zugewiesen wird. Dieses Feld kann bearbeitet werden.
Signature	Hier geben Sie die Signatur (Name einer Operation oder eines Signals) der Nachricht an. Wenn Sie die Signatur einer Antwortnachricht ändern, wird auch die Signatur der zugehörigen Aufrufnachricht geändert.
Arguments	Die Argumente einer Operation, die der Nachricht zugeordnet ist. Dieses Feld kann bearbeitet werden. Wenn Sie die Argumente einer Antwortnachricht ändern, wird auch die zugehörige Aufrufnachricht geändert.
Return value	Der Rückgabewert einer Operation, die der Nachrichtenbeziehung zugeordnet ist. Dieses Feld kann bearbeitet werden.
Sort	Hier wählen Sie in einer Dropdown-Liste die Art der Synchronisierung aus. Mögliche Werte sind asynchCall, synchCall und asynchSignal. Die Nachrichtenbeziehung wird entsprechend aktualisiert.
Commentary	Hier geben Sie einen Kommentar für die Beziehung ein.

Anmerkung: Bestimmte Eigenschaften von Aufruf- und Antwortnachrichten, wie z. B. *arguments*, *attribute*, *qualified name*, *return value*, *signature* und *sort* beziehen sich auf die Aufrufspezifikation. Sie können diese Eigenschaften direkt in der Aufrufspezifikation, in der Aufruf- oder in der Antwortnachricht bearbeiten. Die entsprechenden Eigenschaften der anderen Elemente werden dann automatisch aktualisiert. Die Eigenschaften *stereotype* und *commentary* betreffen nur die jeweilige Nachricht.

Siehe auch

Arbeiten mit Nachrichten in UML 1.5 (siehe Seite 158)

Ausführungs- und Aufrufspezifikationen (siehe Seite 1320)

UML 2.0-Interaktionsdiagramm (siehe Seite 1315)

4.2.7.3.7 Ausführungs- und Aufrufspezifikationen

In einem Sequenzdiagramm zeigt Together die Ausführungsspezifikation einer Nachricht mit deren Aktivitätsdauer automatisch an. Wenn Sie eine Nachrichtenbeziehung zur Ziellebenslinie herstellen, wird die Ausführungsspezifikationsleiste automatisch erzeugt. Sie können die Aktivitätsdauer einer Nachricht verlängern oder verkürzen, indem Sie die obere bzw. untere Linie der Spezifikationsleiste nach oben oder unten ziehen.

Sie können auch eine Ausführungsspezifikation auf einer Lebenslinie ohne Nachrichtenbeziehung erstellen. In diesem Fall wird eine gefundene Nachricht erstellt. Dies ist eine Nachricht von einem Objekt, das nicht im Diagramm angezeigt wird. Sie können die gefundenen Nachrichten im des Objektinspektors ein- und ausblenden.

Die Nachrichten in einem Sequenzdiagramm haben ihren Ursprung in einer Aufrufspezifikation. Dies ist ein spezieller Bereich in einer Ausführungsspezifikation. Aufrufspezifikationen werden in der Together-Implementierung der UML 2.0-Sequenzdiagramme erstmals eingeführt. Dieses Element ist zwar nicht in UML 2.0 definiert, es ist aber bei der Modellierung von mit Antworten synchronen Aufrufen hilfreich. Die Aufrufspezifikation ist insbesondere die Stelle im Diagramm, an der die Antworten (auch nicht angezeigte) in den Ausführungskontext einer Lebenslinie eintreten und Unternachrichten wieder auf die Lebenslinie gelangen können.

Die aktiven und passiven Bereiche der Aufrufspezifikation werden in unterschiedlichen Farben angezeigt. Die aktiven Bereiche, in denen Sie Nachrichtenbeziehungen erstellen können, sind weiß unterlegt. Die grauen Bereiche sind passiv und nicht als Quelle oder Ziel von Nachrichtenbeziehungen zulässig.

Siehe auch

UML 2.0-Nachricht (siehe Seite 1318)

4.2.7.3.8 Operator und Operand für ein kombiniertes Fragment

Kombiniertes Fragment

Ein kombiniertes Fragment kann aus einem oder mehreren Interaktionsoperatoren und einem oder mehreren Interaktionsoperanden bestehen. Die Anzahl der Interaktionsoperanden (lediglich einer oder mehrere) richtet sich nach dem letzten Interaktionsoperator des kombinierten Fragments.

Diese Elemente können über die der Tool-Palette erstellt werden. Der Operatortyp wird im Deskriptor links oben im Designelement angezeigt. In einem kombinierten Fragment können mehrere Operatoren definiert werden. In diesem Fall werden alle Operatoren im Deskriptor angezeigt.

Fügen Sie beim Erstellen eines Operators die zulässigen Operanden über das Kontextmenü des kombinierten Fragments hinzu.

Ein kombiniertes Fragment kann auf mehrere Lebenslinien erweitert, von Lebenslinien gelöst und wieder an ihnen verankert werden. Über das Feld *Operatoren* im des Objektinspektors können Sie die Operatoren des kombinierten Fragments verwalten.

Die kombinierten Fragmente sind mit einem schwarzen Punkt an ihren Lebenslinien verankert. Bei diesem Punkt handelt es sich um ein separates Diagrammelement, das ausgewählt oder gelöscht werden kann. Wenn Sie einen Punkt löschen, wird das kombinierte Fragment von der Lebenslinie gelöst. Beachten Sie, dass ein kombiniertes Fragment nicht von sämtlichen Lebenslinien gelöst werden kann und mindestens über einen Punkt verfügen muss.

Kombinierte Fragmente können mit Hilfe des Tools *Frame verankern* jederzeit wieder mit einer Lebenslinie verbunden werden.

Operator

Wenn Sie ein kombiniertes Fragment erstellen, wird der Operator links oben in einem fünfeckigen Deskriptor angezeigt. Sie können den Operatortyp im des Objektinspektors mit dem Feld *Operatoren* ändern. Der neue Operator wird sofort im Deskriptor angezeigt.

Der Deskriptor kann auch mehrere Operatoren enthalten. UML 2.0 sieht diese Notation für verschachtelte kombinierte

Fragmente vor. Sie können in Together diese Notation verwenden oder verschachtelte kombinierte Fragmentknoten erstellen.

Operanden

Operanden werden in einem kombinierten Fragment als Rechtecke angezeigt, die durch gestrichelte Linien voneinander getrennt sind. Wenn Sie ein kombiniertes Fragment erstellen, wird die Anzahl der Operanden durch die Standardeinstellungen des Patterns definiert. Sie können anschließend weitere Operanden hinzufügen oder die vorhandenen entfernen.

Beachten Sie, dass der oberste Bereich des Operators keinen Operanden enthält. Er ist für den Deskriptor reserviert. Wenn Sie auf diesen Bereich klicken, wird der gesamte Operator ausgewählt. Beim Klicken auf eines der gestrichelten Rechtecke wird der betreffende Operand ausgewählt. Wenn ein kombiniertes Fragment lediglich einen Operanden enthält, können Sie dennoch das gesamte Fragment und den Operanden getrennt auswählen.

Siehe auch

Überblick über die OCL-Unterstützung (siehe Seite 1453)

UML 2.0-Interaktionsdiagramme (siehe Seite 1315)

4.2.7.4 UML 2.0-Zustandsmaschinendiagramm

Dieser Abschnitt beschreibt die Elemente eines UML 2.0-Zustandsmaschinendiagramms.

4.2.7.4.1 Definition eines UML 2.0-Zustandsmaschinendiagramms

Zustände sind die Grundeinheiten der Zustandsmaschinen. Sie können in UML 2.0 Unterzustände enthalten.

Die Diagrammausführung beginnt mit dem Anfang-Knoten und hört mit dem Ende- oder Beenden-Knoten auf. Weitere Informationen zu diesen Elementen finden Sie in der UML 2.0-Spezifikation.

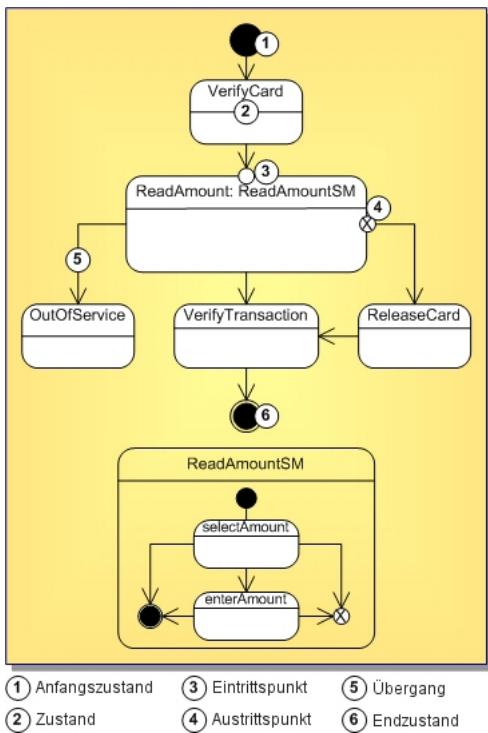
Definition

Ein Zustandsmaschinendiagramm beschreibt das logische Verhalten eines Systems bzw. eines Teils des Systems oder dessen Verwendungsprotokoll.

In diesen Diagrammen werden die möglichen Zustände der Objekte und die Übergänge zu einem anderen Zustand dargestellt.

Die UML 2.0-Zustandsmaschinendiagramme unterscheiden sich in vielerlei Hinsicht von den Zustandsdiagrammen in UML 1.5.

Beispieldiagramm



- ① Anfangszustand ③ Eintrittspunkt ⑤ Übergang
 ② Zustand ④ Austrittspunkt ⑥ Endzustand

4.2.7.4.2 Elemente in einem UML 2.0-Zustandsmaschinendiagramm

In der folgenden Tabelle sind die Elemente eines UML 2.0-Zustandsmaschinendiagramm aufgeführt, die in der der Tool Palette zur Verfügung stehen.

Elemente in einem UML 2.0-Zustandsmaschinendiagramm

Name	Typ	Beschreibung
Status	Knoten	
Eintrittspunkt	Knoten	An diesem Punkt beginnt die Ausführung des Zustands. Für einen Zustand können mehrere Eintrittspunkte erstellt werden. Dies ist sinnvoll, wenn Unterzustände vorhanden sind.
Austrittspunkt	Knoten	An diesem Punkt endet die Ausführung des Zustands. Für einen Zustand können mehrere Austrittspunkte erstellt werden. Dies ist sinnvoll, wenn Unterzustände vorhanden sind.
Initial	Knoten	
Final	Knoten	
Beenden	Knoten	
Flache Historie	Knoten	
Tiefe Historie	Knoten	

Region	Knoten	Mithilfe von Regionen können die Unterzustände in einem Zustand gruppiert werden. Die Regionen können über unterschiedliche Sichtbarkeitseinstellungen und Historienelemente verfügen. In jedem neuen Zustand wird automatisch eine Region erstellt (die gelöscht werden kann). In den Regionen können alle Elemente eines Zustandsmaschinendiagramms erstellt werden. Interne Übergänge können nur über das Kontextmenü eines Zustands erstellt werden.
Aufspaltung	Knoten	
Zusammenführung	Knoten	
Auswahl	Knoten	
Schnittpunkt	Knoten	
Übergang	Beziehung	Erstellen Sie eine Beziehung vom Austrittspunkt des Quellzustands (bzw. vom Zustand, wenn kein Austrittspunkt vorhanden ist) zum Eintrittspunkt des Zielzustands (bzw. zum Zustand, wenn kein Eintrittspunkt vorhanden ist).
Interner Übergang	Beziehung	Interne Übergänge können nur über das Kontextmenü eines Zustands erstellt werden.
Knoten Pattern	nach Öffnet den Pattern-Experten ( siehe Seite 1277).	
Beziehung Pattern	nach Öffnet den Pattern-Experten ( siehe Seite 1277).	
Hinweis	Annotation	
Hinweisbeziehung	Annotationsbeziehung	

4.2.7.4.3 Historienelement (Zustandsmaschinendiagramm)

Die Elemente *Flache Historie* und *Tiefe Historie* können in bestimmte Regionen von Zuständen eingefügt werden.

In jeder Region können sich keine oder eine tiefe Historie und keine oder eine flache Historie befinden. Wenn in einer Region nur ein Historienelement vorhanden ist, kann über die Eigenschaft *kind* zwischen den beiden Historienarten gewechselt werden.

Weitere Informationen zu diesen Elementen finden Sie in der UML 2.0-Spezifikation.

Siehe auch

Erstellen einer Historie ( siehe Seite 188)

Zustandsmaschinendiagramm ( siehe Seite 1321)

4.2.7.5 UML 2.0-Aktivitätsdiagramm

Dieser Abschnitt beschreibt die Elemente eines UML 2.0-Aktivitätsdiagramms.

4.2.7.5.1 Definition eines UML 2.0-Aktivitätsdiagramms

Definition

Mit Hilfe eines Aktivitätsdiagramms können Sie das Systemverhalten, einschließlich der Ausführungsreihenfolge und -bedingungen der Aktionen, modellieren. Aktionen sind die Grundeinheiten des Systemverhaltens.

In einem Aktivitätsdiagramm können Sie Aktionen gruppieren und aufteilen. Wenn eine Aktion in eine Folge anderer Aktionen aufgeteilt wird, können Sie für diese eine Aktivität erstellen.

In UML 2.0 bestehen Aktivitäten aus Aktionen. Aktionen sind keine Zustände (verglichen mit UML 1.x) und können Unteraktionen enthalten. Eine Aktion steht für einen einzelnen Schritt einer Aktivität, d. h. für einen Schritt, der innerhalb der Aktivität nicht weiter zerlegt wird. Eine Aktivität steht für ein Verhalten, das sich aus einzelnen Elementen (Aktionen) zusammensetzt. Eine Aktion ist ein ausführbarer Aktivitätsknoten, der die grundlegende Einheit der ausführbaren Funktionalität in einer Aktion darstellt, im Gegensatz zur Steuerung und dem Datenfluss zwischen Aktionen. Die Ausführung einer Aktion stellt einen Übergang oder eine Verarbeitung im modellierten System dar, unabhängig davon, ob es sich dabei um ein Computer- oder ein anderes System handelt.

Die Semantik der Aktivitäten basiert auf dem Ablauf der Token. Der Begriff "Ablauf" bedeutet, dass die Ausführung eines Knotens die Ausführung anderer Knoten beeinflusst und umgekehrt. Derartige Abhängigkeiten werden im Aktivitätsdiagramm durch Umrandungen dargestellt. Daten- und Kontrollfluss sind in UML 2.0 unterschiedlich.

Ein Kontrollfluss kann mehrere Quellaktionen (d. h. er ist eine Zusammenführung mehrerer gleichzeitiger Aktionen) oder mehrere Zielaktionen haben (d. h. er ist eine Aufspaltung in mehrere gleichzeitige Aktionen).

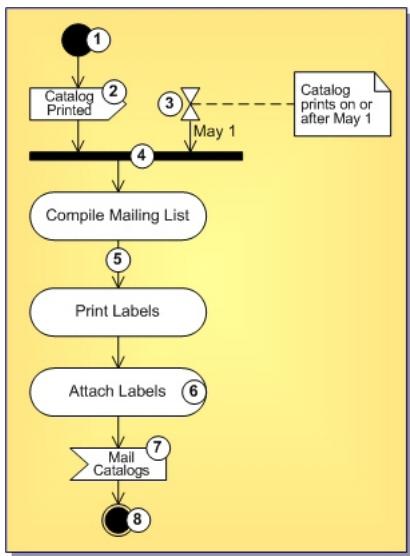
Jede Flussbeziehung in einer Aktivität hat ihr eigenes Ende, das durch einen Flussendeknoten angegeben wird. Dieser Knoten bedeutet, dass ein bestimmter Ablauf in einer Aktivität abgeschlossen ist. Beachten Sie, dass vom Flussende keine Beziehungen ausgehen können.

Mit Hilfe von Entscheidungen und Zusammenfassungen können Sie mehrere aus- oder eingehende Kontrollflussbeziehungen verwalten.

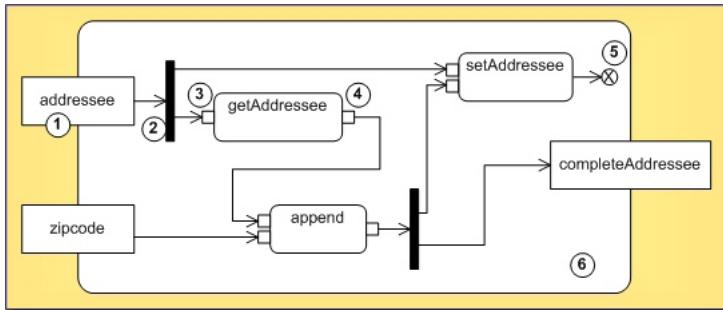
Im Objektinspektors können Sie Aktionseigenschaften einstellen.

- Konfigurieren Sie auf den Registerkarten Eigenschaften, Ansicht, Beschreibung und Benutzerdefiniert die Standardeigenschaften des Elements.
- Wählen Sie auf den Registerkarten Lokale Vorbedingung und Lokale Nachbedingung die Sprache des Einschränkungsausdrucks in der Liste Sprache aus. Die möglichen Optionen sind OCL und Text. Geben Sie den Ausdruck in das Textfeld unter der Liste ein.

Beispieldiagramm



- | | |
|-------------------------------------|---------------------------------|
| (1) Anfangszustand | (5) Kontrollablauf |
| (2) Aktion Signal senden | (6) Aktion |
| (3) Aktion Zeitereignis akzeptieren | (7) Aktion Ereignis akzeptieren |
| (4) Zusammenföhrung | (8) Endzustand |



- | | | |
|-------------------------|-----------------|----------------|
| (1) Aktivitätsparameter | (3) Eingabe-Pin | (5) Ablaufende |
| (2) Aufspaltung | (4) Ausgabe-Pin | (6) Aktivität |

4.2.7.5.2 Elemente in einem UML 2.0-Aktivitätsdiagramm

In der folgenden Tabelle sind die Elemente eines UML 2.0-Aktivitätsdiagramms aufgeführt, die in der der Tool-Palette zur Verfügung stehen.

Elemente in einem UML 2.0-Aktivitätsdiagramm

Name	Typ
Aktivität	Knoten
Aktivitätsparameter	Knotenkomponente
Aktion	Knoten
Initial	Knoten
Aktivitätsende	Knoten

4

Entscheidung	Knoten
Zusammenfassung	Knoten
Ablaufende	Knoten
Kontrollablauf	Beziehung
Eingabe-Pin	Pin
Ausgabe-Pin	Pin
Wert-Pin	Pin
Objektknoten	Knoten
Zentraler Puffer	Knoten
Datenspeicherung	Knoten
Objektfluss	Beziehung
Aktion Ereignis akzeptieren	Knoten
Aktion Zeitereignis akzeptieren	Knoten
Aktion Signal senden	Knoten
Knoten nach Pattern	Öffnet den Pattern-Experten (siehe Seite 1277).
Beziehung nach Pattern	Öffnet den Pattern-Experten (siehe Seite 1277).
Hinweis	Annotation
Hinweisbeziehung	Annotationsbeziehung

4.2.7.5.3 Pins

Viele **Aktionen** benötigen Eingabewerte oder produzieren eine Ausgabe. Eingabe-, Ausgabe- und Wert-**Pins** enthalten solche Werte.

Siehe auch

Erstellen von Eingabe- (siehe Seite 161)

Referenz eines UML 2.0-Aktivitätsdiagramms (siehe Seite 1323)

4.2.7.6 UML 2.0-Komponentendiagramm

Dieser Abschnitt beschreibt die Elemente eines UML 2.0-Komponentendiagramms.

4.2.7.6.1 Definition eines UML 2.0-Komponentendiagramms

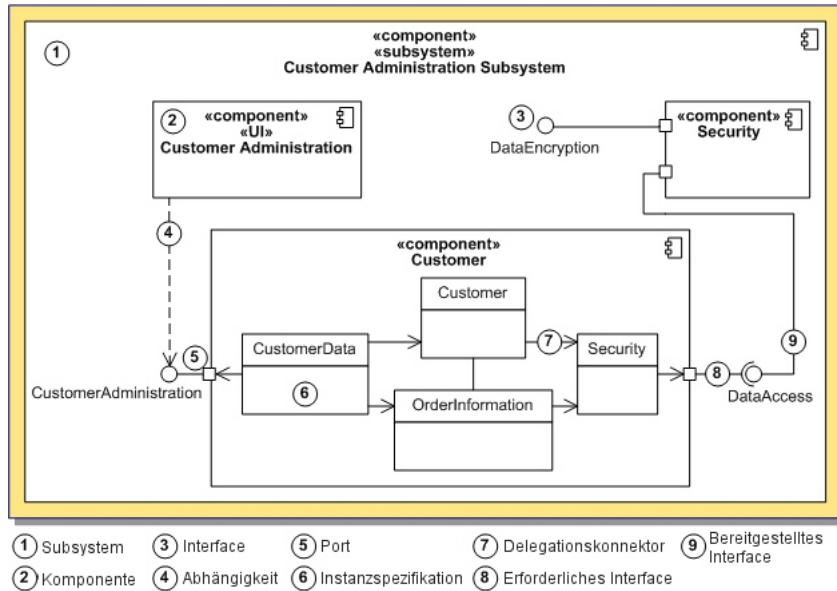
Dieses Thema beschreibt das UML 2.0-Komponentendiagramm.

Definition

Gemäß dem UML 2.0-Standard kann ein Komponentendiagramm Instanzspezifikationen enthalten. Eine Instanzspezifikation kann durch einen oder mehrere Klassifizierer definiert werden. Als Klassifizierer können Klassen, Schnittstellen oder Komponenten verwendet werden. Sie können einen Klassifizierer über das Objektinspektors oder über den internen Editor instantiiieren.

Beispieldiagramm

Mit Hilfe von Komponentendiagrammen können Softwaresysteme beliebiger Größe und Komplexität beschrieben werden.



4.2.7.6.2 Elemente in einem UML 2.0-Komponentendiagramm

In der folgenden Tabelle sind die Elemente eines UML 2.0-Komponentendiagramms aufgeführt, die in der der Tool Palette zur Verfügung stehen.

Elemente in einem UML 2.0-Komponentendiagramm

Name	Typ
Komponente	Knoten
Klasse	Knoten
Port	Knoten
Artefakt	Knoten
Interface	Knoten
Instanzspezifikation	Knoten
Delegationskonnektor	Beziehung
Bereitgestelltes Interface	Beziehung
Erforderliches Interface	Beziehung
Assoziation	Beziehung
Aggregation	Beziehung
Abhangigkeit	Beziehung
Realisierung	Beziehung
Knoten nach Pattern	Offnet den Pattern-Experten (siehe Seite 1277).
Beziehung nach Pattern	Offnet den Pattern-Experten (siehe Seite 1277).
Hinweis	Annotation

Hinweisbeziehung

Annotationsbeziehung

4.2.7.7 UML 2.0-Verteilungsdiagramm

Dieser Abschnitt beschreibt die Elemente eines UML 2.0-Verteilungsdiagramms.

4.2.7.7.1 Definition eines UML 2.0-Verteilungsdiagramms

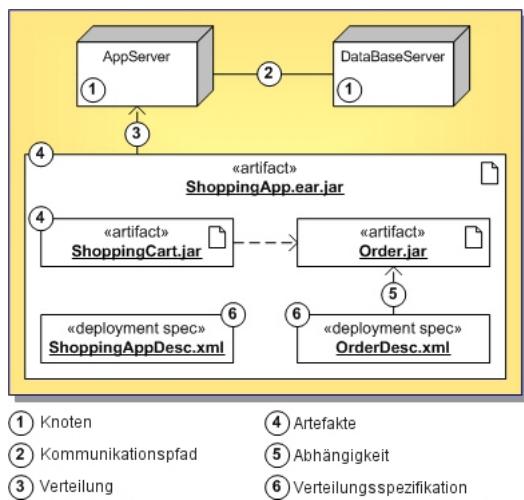
Dieses Thema beschreibt das UML 2.0-Verteilungsdiagramm.

Definition

Ein Verteilungsdiagramm definiert die Elemente, mit denen die Ausführungsarchitektur von Systemen zur Zuweisung von Softwarekomponenten zu Knoten beschrieben wird. Mit Hilfe von Kommunikationspfaden zwischen den Knoten können Netzwerksysteme beliebiger Komplexität modelliert werden. Die Knoten werden in der Regel verschachtelt und stellen entweder Hardwaregeräte oder softwareseitige Ausführungsumgebungen dar. Artefakte entsprechen den konkreten physischen Elementen, die das Ergebnis eines Entwicklungsprozesses sind.

Das Diagramm wurde folgendem Buch entnommen: *Unified Modeling Language: Superstructure Version 2.0. vom August 2003, S. 207, 212.*

Beispieldiagramm



4.2.7.7.2 Elemente in einem UML 2.0-Verteilungsdiagramm

In der folgenden Tabelle sind die Elemente eines UML 2.0-Verteilungsdiagramms aufgeführt, die in der der Tool Palette zur Verfügung stehen.

Elemente in einem UML 2.0-Verteilungsdiagramm

Name	Typ
Knoten	Knoten
Artefakt	Knoten
Gerät	Knoten

Ausführungsspezifikation	Knoten
Verteilungsspezifikation	Knoten
Instanzspezifikation	Knoten
Verteilung	Beziehung
Generalisierung	Beziehung
Assoziation	Beziehung
Abhängigkeit	Beziehung
Manifestation	Beziehung
Kommunikationspfad	Beziehung
Knoten nach Pattern	Öffnet den Pattern-Experten (siehe Seite 1277).
Beziehung nach Pattern	Öffnet den Pattern-Experten (siehe Seite 1277).
Hinweis	Annotation
Hinweisbeziehung	Annotationsbeziehung

Ein Artefakt entspricht einer physischen Entität und wird im Diagramm als Rechteck mit dem Stereotyp <<artifact>> angezeigt. Artefakte können über Eigenschaften, die ihre Funktionsweise definieren, und Operationen, die mit ihren Instanzen ausgeführt werden, verfügen. Beispiele für Artefakte sind Modelldateien, Quelltextdateien, Skripts, ausführbare Dateien, Datenbanktabellen, E-Mails, Textverarbeitungsdokumente usw. Verteilte Artefakte sind Artefakte, die an einen als Verteilungsziel verwendeten Knoten weitergegeben wurden. Sie werden durch Verteilungsbeziehungen mit dem Zielknoten verbunden.

Artefakte können Operationen enthalten.

Sie können komplexe Artefakte erstellen, indem Sie die Artefaktnoten verschachteln.

4.2.7.8 UML 2.0-Kompositionsstrukturdiagramm

Dieser Abschnitt beschreibt die Elemente eines UML 2.0-Kompositionsstrukturdiagramms.

4.2.7.8.1 Definition eines UML 2.0-Kompositionsstrukturdiagramms

Das Diagramm wurde folgendem Buch entnommen: *Unified Modeling Language: Superstructure Version 2.0. vom August 2003 (Seite 178)*.

Definition

Ein Kompositionsstrukturdiagramm zeigt die interne Struktur eines Klassifizierers, einschließlich der Interaktionspunkte zu den anderen Elementen des Systems. Es stellt die Konfiguration der Elemente dar, die zusammen das Verhalten des Klassifizierers bestimmen.

Eine Kollaboration beschreibt die Struktur der kollaborierenden Parts (Rollen). Kollaborationen werden Operationen oder Klassifizierern über ein Kollaborationsverwendungselement hinzugefügt.

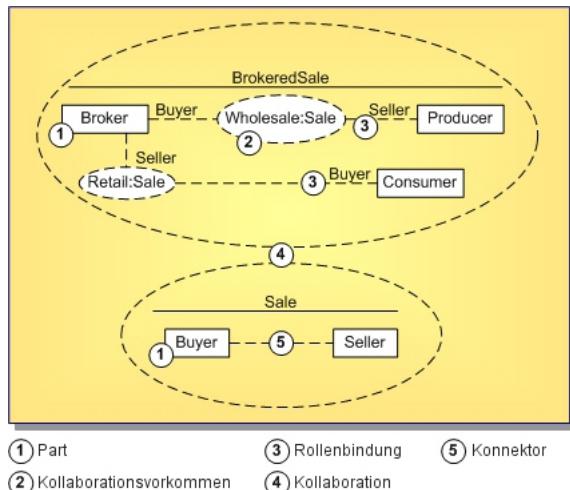
Die Klassen und Kollaborationen in einem Kompositionsstrukturdiagramm können eine interne Struktur und Ports enthalten. Die interne Struktur wird durch eine Gruppe untereinander verbundener Parts (Rollen) definiert. Die Beziehungen zwischen den Teilnehmern einer Kollaboration oder Klasse werden durch Konnektoren hergestellt.

Ein Port kann entweder auf einem enthaltenen Part oder an der Klassengrenze auftreten.

Die Parts können auch als Referenzen eingefügt werden. Sie werden dann durch gestrichelte Rechtecke angezeigt.

Kompositionssstrukturdiagramme unterstützen die Kugelgelenksnotation für die bereitgestellten und erforderlichen Interfaces. Die Interfaces können bei Bedarf im Diagramm ein- oder ausgeblendet werden.

Beispieldiagramm



4.2.7.8.2 Elemente in einem UML 2.0-Kompositionssstrukturdiagramm

In der folgenden Tabelle sind die Elemente eines UML 2.0-Kompositionssstrukturdiagramms aufgeführt, die in der der Tool Palette zur Verfügung stehen.

Elemente in einem UML 2.0-Kompositionssstrukturdiagramm

Name	Typ
Klasse	Knoten
Interface	Knoten
Kollaboration	Knoten
Kollaborationsverwendung	Knoten
Part	Knoten
Referenziertes Part	Knoten
Port	Knoten
Bereitgestelltes Interface	Beziehung
Erforderliches Interface	Beziehung
Konnektor	Beziehung
Kollaborationsrolle	Beziehung
Rollenbindung	Beziehung
Knoten nach Pattern	Öffnet den Pattern-Experten (siehe Seite 1277).
Beziehung nach Pattern	Öffnet den Pattern-Experten (siehe Seite 1277).
Hinweis	Annotation
Hinweisbeziehung	Annotationsbeziehung

4.2.7.8.3 Delegationskonnektor

Ein Interface kann ihre Verpflichtungen über einen Delegationskonnektor auf ein anderes Interface übertragen.

Siehe auch

Referenz eines UML 2.0-Kompositionsstrukturdiagramms (siehe Seite 1329)

4.2.8 Together – Refactoring-Operationen

Together unterstützt die folgenden Refactoring-Operationen:

Refactoring-Operationen

Operation	Beschreibung
Parameter ändern	Sie können Parameter für eine einzelne Methode in der Modellansicht, in der Diagrammansicht oder im Editor umbenennen, hinzufügen oder entfernen.
Interface extrahieren	Mit dem Befehl Interface extrahieren wird ein neues Interface aus einer oder mehreren ausgewählten Klassen erstellt. Jede ausgewählte Klasse sollte das Interface implementieren.
Methode extrahieren	Das Extrahieren einer Methode aus einer Klasse oder einem Interface ist nur im Editor möglich. Das zu extrahierende Quelltextfragment muss vollständige Anweisungen enthalten.
Superklasse extrahieren	Mit dem Befehl Superklasse extrahieren kann eine Vorfahrenklasse aus mehreren Membern einer bestimmten Klasse erstellt werden.
Inline für Variable	Wenn Sie einer temporären Variablen einen einfachen Ausdruck zugewiesen haben, können Sie alle Referenzen auf diese Variable durch den Ausdruck ersetzen lassen. Dazu steht im RAD Studio-Editor der Befehl Inline für Variable zur Verfügung.
Feld einführen	Ein neues Feld wird erstellt.
Variable einführen	Eine neue Variable wird erstellt. Dieser Befehl steht im Editor zur Verfügung.
Member verlagern	Mit diesem Befehl können statische Methoden, statische Felder und statische Eigenschaften (statische Member) verschoben werden. Er steht in der Diagrammansicht, in der Modellansicht und im Editor zur Verfügung.
Member in übergeordnete Klasse verschieben	Mit diesem Befehl können Sie ein Member (Feld, Methode, Eigenschaft, Indizierer oder Ereignis) aus einer Subklasse in eine Superklasse verlagern und optional abstrakt machen. Wenn keine Superklassen vorhanden sind, wird eine Fehlermeldung angezeigt.
Member in abgeleitete Klasse verschieben	Mit diesem Befehl können Sie ein Member (Feld, Methode, Eigenschaft oder Ereignis) aus einer Superklasse in eine Subklasse verlagern und optional abstrakt machen. Wenn keine Subklassen vorhanden sind, wird eine Fehlermeldung angezeigt. Indizierer können nicht in abgeleitete Klassen verschoben werden.
Sicheres Löschen	Mit dem Befehl Sicheres Löschen können Sie den Quelltext nach Referenzen auf das Element durchsuchen, das Sie löschen möchten. Der Befehl kann in der Diagrammansicht, der Modellansicht oder im Editor aufgerufen werden.

Siehe auch

- Überblick zum Refactoring (siehe Seite 1456)
Refactoring-Operationen verwenden (siehe Seite 1401)

4.2.9 Projekttypen und -formate für das Modellieren

Es gibt zwei grundlegenden Projekttypen:

- **Designprojekte.** Erweiterung der Projektdatei: `.bdsproj.tgproj`. Diese Projekte sind sprachneutral und entsprechen einer der beiden Versionen der UML-Spezifikation: UML 1.5 oder UML 2.0.
- **Implementierungsprojekte:** Erweiterung der Projektdatei: `.bdsproj`. Sie können Modelle für sprachspezifische Projekte erstellen. Die Modellierung nach der UML 1.5-Spezifikation wird für C#- und Delphi-Projekte unterstützt. Für diese Projekte werden die Together-Modellierungsfunktionen automatisch aktiviert.

Together unterstützt für die oben genannten Projekttypen verschiedene Projektformate:

Unterstützte Projektformate

Projektformat	Projekttyp	Unterstützte Standardoperationen
Delphi-Formate	Implementierung	Erstellen, Öffnen, Speichern, Bearbeiten
C# .NET	Implementierung	Erstellen, Öffnen, Speichern, Bearbeiten
Together-Designformate: UML 1.5, UML 2.0	Design	Erstellen, Öffnen, Speichern, Bearbeiten
Andere Editionen von Together	Design oder Implementierung	Import, Freigabe
IBM Rational Rose-Format (MDL)	Design	Erstellen eines neuen Designprojekts mit dem Importexperten
XMI-Format	Design	Import, Export

Siehe auch

- Überblick zu Modellierungsprojekten (siehe Seite 1446)
Überblick zur Interoperabilität (siehe Seite 1460)
Designprojekte in Quelltext umwandeln – Überblick (siehe Seite 1452)
Ein Projekt erstellen (siehe Seite 249)

4.3 Befehlszeilenoptionen

Im folgenden Thema werden die Befehlszeilenoptionen der IDE beschrieben.

4.3.1 IDE-Befehlszeilenoptionen

Beschreibung der Optionen für den Start der IDE von der Befehlszeile.

IDE-Befehlszeilenoptionen

Im Folgenden werden die Optionen beschrieben, die Sie beim Start der IDE von der Befehlszeile angeben können. Allen Optionen muss entweder ein Bindestrich (-) oder ein Schrägstrich (/) vorangestellt werden (falls nicht anders angegeben). Da keine Unterscheidung zwischen Groß- und Kleinschreibung stattfindet, sind folgende Optionen identisch: -d /d -D /D.

Die Optionen werden mit dem Befehl zum Start der IDE eingegeben: bds .exe

Zum Beispiel:

Code	Ergebnis
bds.exe -ns	Startet die IDE ohne Eingangsbildschirm.
bds.exe -sdc:\test\source -d c:\test\myprog.exe -td	Lädt beim Start der IDE die Datei C:\TEST\MYPROG.EXE in den Debugger und legt C:\TEST\SOURCE als Verzeichnis für den Quelltext fest. Die Option -td und alle anderen Argumente nach der Debugger-Option -dExeName werden als Parameter an die ausführbare Datei C:\TEST\MYPROG.EXE übergeben.

Allgemeine Optionen

Option	Beschreibung
?	Öffnet die IDE und aktiviert die Online-Hilfe für die IDE-Befehlszeilenoptionen.
	Der Rest der Befehlszeile wird ignoriert.
ns	Kein Startbildschirm. Die IDE wird ohne Eingangsbildschirm gestartet.
np	Keine Willkommensseite. Die Willkommensseite wird nach dem Start der IDE nicht angezeigt.
rregkey	Es kann ein anderer Basis-Registrierungsschlüssel angegeben werden, sodass die Ausführung zweier Instanzen der IDE mit unterschiedlichen Konfigurationen möglich ist. Komponentenentwickler können dann zum Debuggen einer Komponente während des Entwurfs die IDE als Hostanwendung verwenden, ohne dass die Debug-IDE durch das Laden des Komponenten-Package Konflikte verursacht.

Debugger-Optionen

Option	Beschreibung
attach:%1;%2	Eine Debugging-Zuordnung wird vorgenommen. Das Argument %1 wird als Prozess-ID, %2 als Ereignis-ID für diesen Prozess verwendet. Obwohl die Funktion auch manuell durchgeführt werden kann, wird sie meistens beim Just-in-Time-Debugging eingesetzt.

dExeName	Die angegebene ausführbare Datei (ExeName) wird in den Debugger geladen. Alle nach ExeName angegebenen Optionen werden von der IDE ignoriert und als Parameter an das Programm übergeben. Zwischen dem "d" und ExeName kann ein Leerzeichen stehen.
----------	---

Die folgenden Optionen können nur zusammen mit der Option **-d** verwendet werden:

debugger=[borwin32 bordotnet]	Legt fest, welcher Debugger verwendet werden soll. Borwin32 ruft den Borland Win32 Debugger auf. Bordotnet ruft den Borland .NET Debugger auf. Wird diese Option weggelassen, wird der Debugger verwendet, der zuerst bei der IDE registriert wurde. Sie können diese speziellen Option sowohl mit der Option attach als auch mit der Option -d verwenden.
l	(kleines L) Assembler-Start. Der Startcode wird nicht ausgeführt. Diese Option muss zusammen mit "d" verwendet werden. Sie bewirkt, dass der Startcode nicht ausgeführt wird. Normalerweise versucht der Debugger, den Prozess bis zur Funktion main oder WinMain auszuführen. Mit dieser Option wird der Prozess lediglich geladen, aber kein Startcode ausgeführt.
sdVerzeichnisse	Quellverzeichnisse. Diese Option muss zusammen mit -d verwendet werden. Sie legt das oder die Quelltextverzeichnisse fest. Mehrere Werte müssen jeweils durch ein Semikolon getrennt werden. Die Verzeichnisse werden für die Einstellung Pfad für Debugger verwendet (die auch in der Registerkarte Projekt>Optionen>Debugger festgelegt werden kann). Zwischen "sd" und der Verzeichnisliste darf kein Leerzeichen stehen.
hHost-Name	Host-Name des externen Debuggers. Diese Option muss zusammen mit -d verwendet werden. Eine externe Debug-Sitzung wird gestartet, wobei der Host-Name den Host angibt, auf dem das Debuggen ausgeführt wird. Das externe Debug-Server-Programm muss auf dem externen Host laufen.
tArbeitsverzeichnis	Legt das Arbeitsverzeichnis für Ihre Debug-Sitzung fest (entspricht der Einstellung Arbeitsverzeichnis im Dialogfeld Prozess laden).

Projektoptionen

Option	Beschreibung
Dateiname	Die (ohne Bindestrich) angegebene Datei wird in die IDE geladen. Dies kann ein Projekt, eine Projektgruppe oder eine bestimmte Einzeldatei sein.
b	AutoBuild. Diese Option muss zusammen mit "Dateiname" verwendet werden. Sie bewirkt, dass das angegebene Projekt bzw. die Projektgruppe beim Start der IDE automatisch erstellt wird. Die dabei generierten Hinweise, Fehler und Warnungen werden in einer Datei gespeichert. Danach wird die IDE geschlossen. Auf diese Weise können Anwendungen im Batch-Modus mit Hilfe einer Batch-Datei erstellt werden. Die Fehlerstufe ist 0 bei erfolgreicher Erstellung und 1 im Fehlerfall. Standardmäßig erhält die Ausgabedatei den Namen "Dateiname" mit der Erweiterung .ERR. Dieses Verhalten kann mit der Option "o" außer Kraft gesetzt werden.
m	AutoMake. Diese Option entspricht "b", führt aber anstelle einer vollständigen Erstellung ein Make durch.
oAusgabedatei	Ausgabedatei. Diese Option muss zusammen mit "b" oder "m" verwendet werden und legt die Ausgabedatei fest. Diese Datei wird anstelle der Standarddatei für Hinweise, Warnungen und Fehlermeldungen verwendet.

4.4 Tastenzuordnungen

Die folgenden Themen behandeln die in RAD Studio verfügbaren Tastaturvorlagen. Verwenden Sie die Seite [Tools>Optionen>Editor-Optionen>Tastaturvorlagen](#), um die Standard-Tastaturvorlage zu ändern.

4.4.1 Standard-Tastaturvorlage

Die folgende Tabelle enthält die Tastenkombinationen der Standard-Tastaturvorlage für den Quelltext-Editor.

Anmerkung: Wenn auf der Seite [Tools>Optionen>Editor-Optionen>Tastaturbelegung](#) die Option Strg+Alt-Tasten verwenden nicht markiert ist, sind die Tastenkombinationen, die Strg+Alt enthalten, deaktiviert.

Tastenkombination	Aktion
Alt+[Sucht passenden Begrenzer (vorwärts).
Alt+]	Sucht passenden Begrenzer (rückwärts)
Alt+Links	Wechselt nach den Operationen Alt+Auf oder Strg+Klick (Zu Deklaration gehen) zurück.
Alt+F7	Wechselt in der Meldungsansicht zum vorherigen Fehler oder zur vorherigen Meldung.
Alt+F8	Wechselt in der Meldungsansicht zum nächsten Fehler oder zur nächsten Meldung.
Alt+Bild Ab	Wechselt zur nächsten Registerkarte.
Alt+Bild Auf	Wechselt zur vorherigen Registerkarte.
Alt+Rechts	Geht nach einer Alt+Links -Operation weiter.
Alt+Umschalt+Unten	Verschiebt den Cursor um eine Zeile nach unten und markiert die Spalte links von der Startposition des Cursors.
Alt+Umschalt+Ende	Markiert die Spalte von der Cursorposition bis zum Ende der aktuellen Zeile.
Alt+Umschalt+Pos1	Markiert die Spalte von der Cursorposition bis zum Zeilenanfang.
Alt+Umschalt+Links	Markiert die Spalte links vom Cursor.
Alt+Umschalt+Bild Ab	Verschiebt den Cursor um einen Bildschirm nach unten und markiert die Spalte links von der Startposition des Cursors.
Alt+Umschalt+Bild Auf	Verschiebt den Cursor um einen Bildschirm nach oben und markiert die Spalte links von der Startposition des Cursors.
Alt+Umschalt+Rechts	Markiert die Spalte rechts vom Cursor.
Alt+Umschalt+Oben	Verschiebt den Cursor um eine Zeile nach oben und markiert die Spalte links von der Startposition des Cursors.
Alt+Auf	Gehe zu Deklaration.
Klick+Alt+Maus	Markiert spaltenorientierte Blöcke.
Strg+/-	Fügt die Kommentarzeichen // vor jeder Zeile eines markierten Codeblocks hinzu oder entfernt diese.
Strg+Alt+F12	Zeigt eine Dropdown-Liste geöffneter Dateien an.

Strg+Alt+Umschalt+Ende	Markiert die Spalte von der Cursorposition bis zum Dateiende.
Strg+Alt+Umschalt+Pos1	Markiert die Spalte von der Cursorposition bis zum Dateianfang.
Strg+Alt+Umschalt+Links	Markiert die Spalte links vom Cursor.
Strg+Alt+Umschalt+Bild Ab	Markiert die Spalte von der Cursorposition bis zum Anfang des Bildschirms.
Strg+Alt+Umschalt+Bild Auf	Markiert die Spalte von der Cursorposition bis zum Ende des Bildschirms.
Strg+Alt+Umschalt+Rechts	Markiert die Spalte rechts vom Cursor.
Strg+Rück	Löscht das Wort Rechts vom Cursor.
Strg+Klick	Gehe zu Deklaration.
Strg+Entf	Löscht den markierten Block.
Strg+Ab	Verschiebt den Text um eine Zeile nach unten.
Strg+Ende	Verschiebt den Cursor ans Ende der Datei.
Strg+Eingabe	Öffnet eine Datei an der Cursorposition.
Strg+Pos1	Setzt den Cursor an den Dateianfang.
Strg+I	Fügt ein Tab-Zeichen ein.
Strg+J	Popup-Menü für Vorlagen.
Strg+K+n	Setzt eine Positionsmarke, wobei <i>n</i> eine Zahl von 0 bis 9 ist.
Strg+K+T	Markiert ein Wort.
Strg+Links	Setzt den Cursor an den Anfang des vorigen Worts.
Strg+n	Springt zu einer Positionsmarke, wobei <i>n</i> die Nummer (0 bis 9) der Positionsmarke ist.
Strg+N	Fügt eine neue Zeile ein.
Strg+O+C	Schaltet die Blockbildung für Spalten ein.
Strg+O+K	Schaltet die Blockbildung für Spalten aus.
Strg+O+L	Schaltet den Blockmodus für Zeilen ein.
Strg+O+O	Fügt Compiler-Optionen ein.
Strg+P	Interpretiert das nächste Zeichen als Folge von ASCII-Code.
Strg+Bild Ab	Verschiebt den Cursor an den unteren Bildschirmrand.
Strg+Bild Auf	Verschiebt den Cursor an den oberen Bildschirmrand.
Strg+Q+#+	Gehe zu Positionsmarke.
Strg+Rechts	Setzt den Cursor an den Anfang des nächsten Worts.
Strg+Umschalt+C	Aktiviert die Klassenvervollständigung für Klassendeklarationen, in welchen sich der Cursor befindet.
Strg+Umschalt+#+	Setzt eine Positionsmarke.
Strg+Umschalt+B	Zeigt die Pufferliste an.
Strg+Umschalt+Ab	Wechselt zwischen Deklaration und Implementierung.
Strg+Umschalt+Eingabe	Sucht Verwendungen.
Strg+Umschalt+J	Aktiviert den Sync-Bearbeitungsmodus.
Strg+Umschalt K+A	Erweitert alle Codeblöcke.
Strg+Umschalt K+C	Blendet alle Klassen aus.

Strg+Umschalt K+E	Blendet einen Codeblock aus.
Strg+Umschalt K+G	Initialisiert/beendet Interface-Implementierung.
Strg+Umschalt K+M	Blendet alle Methoden aus.
Strg+Umschalt K+N	Blendet alle Namespaces/Unit aus.
Strg+Umschalt K+O	Aktiviert bzw. deaktiviert das Code-Folding.
Strg+Umschalt K+P	Blendet alle geschachtelten Prozeduren aus.
Strg+Umschalt K+R	Blendet alle Regionen aus.
Strg+Umschalt K+T	Blendet den aktuellen Codeblock ein bzw. aus.
Strg+Umschalt K+U	Blendet einen Codeblock ein.
Strg+Umschalt+Ende	Markiert den Text zwischen Cursorposition und Dateiende.
Strg+Umschalt+G	Fügt eine neue GUID (Globally Unique Identifier) ein.
Strg+Umschalt+Pos1	Markiert den Text zwischen Cursorposition und Dateianfang.
Strg+Umschalt+I	Rückt einen Block ein.
Strg+Umschalt+Links	Markiert das Wort links vom Cursor.
Strg+Umschalt+P	Führt ein aufgezeichnetes Tastenmakro aus.
Strg+Umschalt+Bild Ab	Markiert den Text zwischen Cursorposition und unterem Bildschirmrand.
Strg+Umschalt+Bild Auf	Markiert den Text zwischen Cursorposition und oberem Bildschirmrand.
Strg+Umschalt+R	Schaltet zwischen Start und Stopp bei der Aufzeichnung eines Tastenmakros um.
Strg+Umschalt+Rechts	Markiert das Wort rechts vom Cursor.
Strg+Umschalt+Leer	Popup-Fenster für Codeparameter.
Strg+Umschalt+T	Erstellt einen TODO-Eintrag.
Strg+Umschalt+Tab	Verschiebt den Cursor auf die vorhergehende Codeseite (oder Datei).
Strg+Umschalt+Tab	Verschiebt den Cursor auf die vorhergehende Seite.
Strg+Umschalt+U	Rückt einen Block aus.
Strg+Umschalt+Links	Wechselt zwischen Deklaration und Implementierung.
Strg+Umschalt+Y	Löscht die Zeichen bis zum Ende der Zeile.
Strg+Leer	Popup-Fenster für Code-Vervollständigung.
Strg+T	Löscht ein Wort.
Strg+Tab	Verschiebt den Cursor auf die nächste Codeseite (oder Datei).
Strg+Auf	Verschiebt den Text um eine Zeile nach oben.
Strg+Y	Löscht eine Zeile.
F1	Zeigt Hilfe für den markierten vollqualifizierten Namespace an.
Umschalt+Alt+Pfeil	Markiert spaltenorientierte Blöcke.
Umschalt+Rück	Löscht das Zeichen links vom Cursor.
Umschalt+Ab	Verschiebt den Cursor um einen Bildschirm nach unten und markiert den übersprungenen Text.
Umschalt+Ende	Markiert Text von der Cursorposition bis zum Ende der aktuellen Zeile.
Umschalt+Eingabe	Fügt eine neue Zeile und ein Wagenrücklaufzeichen ein.
Umschalt+Pos1	Markiert Text von der Cursorposition bis zum Zeilenanfang.
Umschalt+Links	Markiert das Zeichen links vom Cursor.

Umschalt+Bild Ab	Verschiebt den Cursor um einen Bildschirm nach unten und markiert den übersprungenen Text.
Umschalt+Bild Auf	Verschiebt den Cursor um einen Bildschirm nach oben und markiert den Text links von der Startposition des Cursors.
Umschalt+Rechts	Markiert das Zeichen rechts vom Cursor.
Umschalt+Leer	Fügt einen Leerzeichen ein.
Umschalt+Tab	Verschiebt den Cursor zur vorhergehenden Tabposition.
Umschalt+Auf	Verschiebt den Cursor um eine Zeile nach oben und markiert den übersprungenen Text.

4.4.2 Tastaturvorlage IDE - Klassisch

Die folgende Tabelle enthält die Tastenkombinationen der Tastaturvorlage IDE - Klassisch für den Quelltext-Editor.

Anmerkung: Wenn auf der Seite **Tools>Optionen>Editor-Optionen>Tastaturbelegung** die Option Strg+Alt-Tasten verwenden nicht markiert ist, sind die Tastenkombinationen, die Strg+Alt enthalten, deaktiviert.

Tastenkombination	Aktion
Alt + [Sucht passenden Begrenzer (vorwärts).
Alt +]	Sucht passenden Begrenzer (rückwärts).
Alt+Bild Ab	Wechselt zur nächsten Registerkarte.
Alt+Bild Auf	Wechselt zur vorherigen Registerkarte.
Alt+Umschalt+Unten	Verschiebt den Cursor um eine Zeile nach unten und markiert die Spalte links von der Startposition des Cursors.
Alt+Umschalt+Ende	Markiert die Spalte von der Cursorposition bis zum Ende der aktuellen Zeile.
Alt+Umschalt+Pos1	Markiert die Spalte von der Cursorposition bis zum Zeilenanfang.
Alt+Umschalt+Links	Markiert die Spalte links vom Cursor.
Alt+Umschalt+Bild Ab	Verschiebt den Cursor um einen Bildschirm nach unten und markiert die Spalte links von der Startposition des Cursors.
Alt+Umschalt+Bild Auf	Verschiebt den Cursor um einen Bildschirm nach oben und markiert die Spalte links von der Startposition des Cursors.
Alt+Umschalt+Rechts	Markiert die Spalte rechts vom Cursor.
Alt+Umschalt+Oben	Verschiebt den Cursor um eine Zeile nach oben und markiert die Spalte links von der Startposition des Cursors.
Klick+Alt+Maus	Markiert spaltenorientierte Blöcke.
Strg+ /	Fügt die Kommentarzeichen // vor jeder Zeile eines markierten Codeblocks hinzu oder entfernt diese.
Strg+Alt+Umschalt+Ende	Markiert die Spalte von der Cursorposition bis zum Dateiende.
Strg+Alt+Umschalt+Pos1	Markiert die Spalte von der Cursorposition bis zum Dateianfang.
Strg+Alt+Umschalt+Links	Markiert die Spalte links vom Cursor.
Strg+Alt+Umschalt+Bild Ab	Markiert die Spalte von der Cursorposition bis zum Anfang des Bildschirms.

Strg+Alt+Umschalt+Bild Auf	Markiert die Spalte von der Cursorposition bis zum Ende des Bildschirms.
Strg+Alt+Umschalt+Rechts	Markiert die Spalte rechts vom Cursor.
Strg+Rück	Löscht das Wort Rechts vom Cursor.
Strg+Entf	Löscht den markierten Block.
Strg+Abwärtspfeil	Verschiebt den Text um eine Zeile nach unten.
Strg+Ende	Verschiebt den Cursor ans Ende der Datei.
Strg+Eingabe	Öffnet eine Datei an der Cursorposition.
Strg+Pos1	Setzt den Cursor an den Dateianfang.
Strg+I	Fügt ein Tab-Zeichen ein.
Strg+J	Popup-Menü für Vorlagen.
Strg+Links	Setzt den Cursor an den Anfang des vorigen Worts.
Strg+N	Fügt eine neue Zeile ein.
Strg+O+C	Schaltet die Blockbildung für Spalten ein.
Strg+O+K	Schaltet die Blockbildung für Spalten aus.
Strg+O+O	Fügt Compiler-Optionen ein
Strg+P	Interpretiert das nächste Zeichen als Folge von ASCII-Code.
Strg+Bild Ab	Verschiebt den Cursor an den unteren Bildschirmrand.
Strg+Bild Auf	Verschiebt den Cursor an den oberen Bildschirmrand.
Strg+Rechts	Setzt den Cursor an den Anfang des nächsten Worts.
Strg+Umschalt+C	Aktiviert die Klassenvervollständigung für Klassendeklarationen, in welchen sich der Cursor befindet.
Strg+Umschalt K+A	Erweitert alle Codeblöcke.
Strg+Umschalt K+E	Blendet einen Codeblock aus.
Strg+Umschalt K+O	Aktiviert bzw. deaktiviert das Code-Folding.
Strg+Umschalt K+T	Blendet den aktuellen Codeblock ein bzw. aus.
Strg+Umschalt K+U	Blendet einen Codeblock ein.
Strg+Umschalt+Ende	Markiert den Text zwischen Cursorposition und Dateiende.
Strg+Umschalt+G	Fügt eine neue GUID (Globally Unique Identifier) ein.
Strg+Umschalt+Pos1	Markiert den Text zwischen Cursorposition und Dateianfang.
Strg+Umschalt+I	Rückt einen Block ein.
Strg+Umschalt+Links	Markiert das Wort links vom Cursor.
Strg+Umschalt+Bild Ab	Markiert den Text zwischen Cursorposition und unterem Bildschirmrand.
Strg+Umschalt+Bild Auf	Markiert den Text zwischen Cursorposition und oberem Bildschirmrand.
Strg+Umschalt+Rechts	Markiert das Wort rechts vom Cursor.
Strg+Umschalt+Leer	Popup-Fenster für Codeparameter.
Strg+Umschalt+Tab	Verschiebt den Cursor auf die vorhergehende Codeseite (oder Datei).
Strg+Umschalt+Tab	Verschiebt den Cursor auf die vorhergehende Seite.
Strg+Umschalt+U	Rückt einen Block aus.

Strg+Umschalt+Y	Löscht die Zeichen bis zum Ende der Zeile.
Strg+Leer	Popup-Fenster für Code-Vervollständigung.
Strg+T	Löscht ein Wort.
Strg+Tab	Verschiebt den Cursor auf die nächste Codeseite (oder Datei).
Strg+Aufwärtspfeil	Verschiebt den Text um eine Zeile nach oben.
Strg+Y	Löscht eine Zeile.
F1	Zeigt Hilfe für den markierten vollqualifizierten Namespace an.
Umschalt+Alt+Pfeil	Markiert spaltenorientierte Blöcke.
Umschalt+Rück	Löscht das Zeichen links vom Cursor.
Umschalt+Ab	Verschiebt den Cursor um einen Bildschirm nach UNTEN und markiert den übersprungenen Text.
Umschalt+Ende	Markiert Text von der Cursorposition bis zum Ende der aktuellen Zeile.
Umschalt+Eingabe	Fügt eine neue Zeile und ein Wagenrücklaufzeichen ein.
Umschalt+Pos1	Markiert Text von der Cursorposition bis zum Zeilenanfang.
Umschalt+Links	Markiert das Zeichen links vom Cursor.
Umschalt+Bild Ab	Verschiebt den Cursor um einen Bildschirm nach UNTEN und markiert den übersprungenen Text.
Umschalt+Bild Auf	Verschiebt den Cursor um einen Bildschirm nach oben und markiert den Text links von der Startposition des Cursors.
Umschalt+Rechts	Markiert das Zeichen rechts vom Cursor.
Umschalt+Leer	Fügt einen Leerzeichen ein.
Umschalt+Tab	Verschiebt den Cursor zur vorhergehenden Tabposition.
Umschalt+Auf	Verschiebt den Cursor um eine Zeile nach oben und markiert den übersprungenen Text.

4.4.3 Tastaturvorlage BRIEF-Emulation

Die folgende Tabelle enthält die Tastenkombinationen der Tastaturvorlage BRIEF für den Quelltext-Editor.

Tastenkombination	Aktion
Alt+A	Markiert einen nicht eingeschlossenen Block.
Alt+B	Zeigt eine Liste geöffneter Dateien an.
Alt+Rück	Löscht das Wort Rechts vom Cursor.
Alt+C	Markiert den Beginn eines Spaltenblocks.
Alt+D	Löscht eine Zeile.
Alt+F9	Öffnet das Kontextmenü.
Alt-Minus	Geht zur vorherigen Seite.
Alt+I	Schaltet in den Einfügemodus um.
Alt+K	Löscht bis zum Zeilenende.
Alt+L	Markiert eine Zeile.
Alt+M	Markiert einen inklusiven Block.

Alt+N	Zeigt den Inhalt der nächsten Seite an.
Alt+P	Drückt den markierten Block.
Alt+Bild Ab	Wechselt zur nächsten Registerkarte.
Alt+Bild Auf	Wechselt zur vorherigen Registerkarte.
Alt+Q	Interpretiert das nächste Zeichen als Folge von ASCII-Code.
Alt+R	Liest einen Block aus einer Datei ein.
Rück	Löscht das Zeichen links vom Cursor.
Strg+/-	Fügt die Kommentarzeichen // vor jeder Zeile eines markierten Codeblocks hinzu oder entfernt diese.
Strg+- (Bindestrich)	Schließt die aktuelle Seite.
Strg+B	Verschiebt den Cursor ans untere Ende des Fensters.
Strg+Rück	Löscht das Wort links vom Cursor.
Strg+C	Zentriert im Fenster eine Zeile.
Strg+D	Verschiebt den Cursor um einen Bildschirm nach unten.
Strg+E	Verschiebt den Cursor um einen Bildschirm nach oben.
Strg+Eingabe	Fügt eine neue leere Zeile ein.
Strg+F1	Schlüsselwortsuche des Hilfesystems.
Strg+F5	Schaltet die Beachtung der Groß- und Kleinschreibung als Suchkriterium ein oder aus.
Strg+F6	Schaltet die Beachtung regulärer Ausdrücke als Suchkriterium ein oder aus.
Strg+K	Löscht bis Zeilenanfang.
Strg+M	Fügt eine neue Zeile und ein Wagenrücklaufzeichen ein.
Strg+O+A	Öffnet eine Datei an der Cursorposition.
Strg+O+B	Symbol beim Cursor anzeigen.
Strg+O+O	Kehrt die Groß-/Kleinschreibung der Markierung um.
Strg+Q+[Sucht passenden Begrenzer (vorwärts).
Strg+Q+]	Sucht passenden Begrenzer (rückwärts).
Strg+Q+Strg+[Sucht passenden Begrenzer (vorwärts).
Strg+Q+Strg+]	Sucht passenden Begrenzer (rückwärts).
Strg + S	Führt eine inkrementelle Suche durch.
Strg+T	Verschiebt den Cursor ans obere Ende des Fensters.
Strg+Umschalt+C	Aktiviert die Klassenvervollständigung für Klassendeklarationen, in welchen sich der Cursor befindet.
Entf	Löscht an der Cursorposition ein Zeichen oder einen Block.
Eingabe	Fügt eine neue Zeile und ein Wagenrücklaufzeichen ein.
Esc	Verwirft den letzten Befehl an einer Eingabeaufforderung.
Umschalt+Rück	Löscht das Zeichen links vom Cursor.
Umschalt+F4	Ordnet Fenster horizontal an.
Umschalt+F6	Wiederholt die letzte Suchen/Ersetzen-Operation.
Tabulator	Fügt ein Tab-Zeichen ein.

4.4.4 Tastaturvorlage Epsilon-Emulation

Die folgende Tabelle enthält die Tastenkombinationen der Tastaturvorlage Epsilon für den Quelltext-Editor.

Anmerkung: Wenn auf der Seite [Tools>Optionen>Editor-Optionen>Tastaturbelegung](#) die Option Strg+Alt-Tasten verwenden nicht markiert ist, sind die Tastenkombinationen, die Strg+Alt enthalten, deaktiviert.

Tastenkombination	Aktion
Alt+)	Sucht das nächste passende Begrenzungszeichen (Cursor muss sich auf ')', '}' oder ']' befinden).
Alt+?	Kontextsensitive Hilfe anzeigen.
Alt + \	Löscht Leerzeichen und Tabs links und rechts vom Cursor in derselben Zeile.
Alt+Rück	Löscht das Wort links von der aktuellen Cursorposition.
Alt+C	Wandelt den ersten Buchstaben des Wortes nach dem Cursor zu Großschreibung und alle anderen Buchstaben bis zum Wortende zu Kleinschreibung um.
Alt+D	Löscht bis zum Wort rechts vom Cursor.
Alt+Entf	Löscht den gesamten Text im Block zwischen dem Cursor und dem vorherigen Begrenzungszeichen (der Cursor muss sich auf einem der Zeichen ')', '}' oder ']' befinden).
Alt+L	Schreibt das aktuelle Wort klein.
Alt+Umschalt+ /	Kontextsensitive Hilfe anzeigen.
Alt+Umschalt+O	Sucht das nächste passende Begrenzungszeichen (Cursor muss sich auf ')', '}' oder ']' befinden).
Alt+T	Vertauscht die beiden Wörter rechts und links vom Cursor.
Alt+Tab	Rückt die aktuelle Zeile bis an den Text der vorhergehenden Zeile ein.
Alt+U	Wandelt ein markiertes Wort in Großbuchstaben um oder wandelt alle Buchstaben eines Wortes ab der Cursorposition bis zum Wortende in Großbuchstaben um.
Alt+X	Aktiviert den angegebenen Befehl bzw. das Makro.
Rück	Löscht das Zeichen links von der aktuellen Cursorposition.
Strg+ /	Fügt die Kommentarzeichen // vor jeder Zeile eines markierten Codeblocks hinzu oder entfernt diese.
Strg+ _	Kontextsensitive Hilfe anzeigen.
Strg+Alt+ B	Sucht das nächste passende Begrenzungszeichen (Cursor muss sich auf ')', '}' oder ']' befinden).
Strg+Alt+ F	Sucht das letzte passende Begrenzungszeichen (Cursor muss sich auf ')', '}' oder ']' befinden).
Strg+Alt+ H	Löscht das Wort links von der aktuellen Cursorposition.
Strg+Alt+ K	Löscht den gesamten Text im Block zwischen dem Cursor und dem nächsten Begrenzungszeichen (der Cursor muss sich auf einem der Zeichen ')', '}' oder ']' befinden).
Strg+D	Löscht das markierte Zeichen bzw. das Zeichen rechts vom Cursor.
Strg+H	Löscht das Zeichen links von der aktuellen Cursorposition.
Strg+K	Schneidet den Inhalt einer Zeile aus und fügt ihn in die Zwischenablage ein.
Strg+L	Zentriert das aktive Fenster.
Strg+M	Fügt einen Wagenrücklauf ein.

Strg+O	Fügt hinter dem Cursor eine neue Zeile ein.
Strg+Q	Interpretiert das nächste Zeichen als ASCII-Code.
Strg+R	Inkrementelle Suche rückwärts in der aktuellen Datei.
Strg + S	Inkrementelle Suche nach einer über die Tastatur eingegebenen Zeichenkette.
Strg+Umschalt+-	Kontextsensitive Hilfe anzeigen.
Strg+Umschalt+C	Aktiviert die Klassenvervollständigung für Klassendeklarationen, in welchen sich der Cursor befindet.
Strg+T	Vertauscht die beiden Zeichen rechts und links vom Cursor.
Strg+X+,	Zeigt das Symbol beim Cursor an.
Strg+X+0	Löscht den Inhalt des aktuellen Fensters.
Strg+X+Strg+E	Ruft einen Kommandozeilen-Interpreter auf.
Strg+X+Strg+T	Vertauscht die beiden Zeilen rechts und links vom Cursor.
Strg+X+Strg+X	Tauscht die Position des Cursors mit einem Lesezeichen.
Strg+X+I	Fügt den Inhalt einer Datei an der Cursorposition ein.
Strg+X+N	Zeigt das nächste Fenster in der Pufferliste an.
Strg+X+P	Zeigt das vorherige Fenster in der Pufferliste an.
Esc+)	Sucht das nächste passende Begrenzungszeichen (Cursor muss sich auf ')', '}' oder ']' befinden).
Esc+?	Kontextsensitive Hilfe anzeigen.
Esc+\	Löscht Leerzeichen und Tabs links und rechts vom Cursor in derselben Zeile.
Esc+Rück	Löscht das Wort links von der aktuellen Cursorposition.
Esc+C	Wandelt den ersten Buchstaben des Wortes nach dem Cursor zu Großschreibung und alle anderen Buchstaben bis zum Wortende zu Kleinschreibung um.
Esc+Strg+B	Sucht das nächste passende Begrenzungszeichen (Cursor muss sich auf ')', '}' oder ']' befinden).
Esc+Strg+F	Sucht das letzte passende Begrenzungszeichen (Cursor muss sich auf ')', '}' oder ']' befinden).
Esc+Strg+H	Löscht das Wort links von der aktuellen Cursorposition.
Esc+Strg+K	Löscht den gesamten Text im Block zwischen dem Cursor und dem nächsten Begrenzungszeichen (der Cursor muss sich auf einem der Zeichen ')', '}' oder ']' befinden).
Esc+D	Löscht bis zum Wort rechts vom Cursor.
Esc+Entf	Löscht den gesamten Text im Block zwischen dem Cursor und dem vorherigen Begrenzungszeichen (der Cursor muss sich auf einem der Zeichen ')', '}' oder ']' befinden).
Esc+Ende	Zeigt das nächste Fenster in der Pufferliste an.
Esc+Pos1	Zeigt das vorherige Fenster in der Pufferliste an.
Esc+L	Schreibt das aktuelle Wort klein.
Esc+T	Vertauscht die beiden Wörter rechts und links vom Cursor.
Esc+Tab	Rückt die aktuelle Zeile bis an den Text der vorhergehenden Zeile ein.
Esc+U	Wandelt ein markiertes Wort in Großbuchstaben um oder wandelt alle Buchstaben eines Wortes ab der Cursorposition bis zum Wortende in Großbuchstaben um.
Esc+X	Aktiviert den angegebenen Befehl bzw. das Makro.
F2	Aktiviert den angegebenen Befehl bzw. das Makro.

4.4.5 Tastaturvorlage Visual Studio-Emulation

Die folgende Tabelle enthält die Tastenkombinationen der Tastaturvorlage Visual Studio für den Quelltext-Editor.

Anmerkung: Wenn auf der Seite [Tools>Optionen>Editor-Optionen>Tastaturbelegung](#) die Option Strg+Alt-Tasten verwenden nicht markiert ist, sind die Tastenkombinationen, die Strg+Alt enthalten, deaktiviert.

Tastenkombination	Aktion
Alt + [Sucht passenden Begrenzer (vorwärts).
Alt +]	Sucht passenden Begrenzer (rückwärts).
Alt+Rück	Bearbeiten Rückgängig
Alt+F12	Symbol beim Cursor anzeigen (Delphi).
Alt+Bild Ab	Wechselt zur nächsten Registerkarte.
Alt+Bild Auf	Wechselt zur vorherigen Registerkarte.
Alt+Umschalt+Rück	Bearbeiten Widerrufen
Alt+Umschalt+Unten	Verschiebt den Cursor um eine Zeile nach unten und markiert die Spalte links von der Startposition des Cursors.
Alt+Umschalt+Ende	Markiert die Spalte von der Cursorposition bis zum Ende der aktuellen Zeile.
Alt+Umschalt+Pos1	Markiert die Spalte von der Cursorposition bis zum Zeilenanfang.
Alt+Umschalt+Links	Markiert die Spalte links vom Cursor.
Alt+Umschalt+Bild Ab	Verschiebt den Cursor um einen Bildschirm nach unten und markiert die Spalte links von der Startposition des Cursors.
Alt+Umschalt+Bild Auf	Verschiebt den Cursor um einen Bildschirm nach oben und markiert die Spalte links von der Startposition des Cursors.
Alt+Umschalt+Rechts	Markiert die Spalte rechts vom Cursor.
Alt+Umschalt+Oben	Verschiebt den Cursor um eine Zeile nach oben und markiert die Spalte links von der Startposition des Cursors.
Strg+Alt+Umschalt+Ende	Markiert die Spalte von der Cursorposition bis zum Dateiende.
Strg+Alt+Umschalt+Pos1	Markiert die Spalte von der Cursorposition bis zum Dateianfang.
Strg+Alt+Umschalt+Links	Markiert die Spalte links vom Cursor.
Strg+Alt+Umschalt+Bild Ab	Markiert die Spalte von der Cursorposition bis zum Anfang des Bildschirms.
Strg+Alt+Umschalt+Bild Auf	Markiert die Spalte von der Cursorposition bis zum Ende des Bildschirms.
Strg+Alt+Umschalt+Rechts	Markiert die Spalte rechts vom Cursor.
Strg+F4	Schließt die aktuelle Seite.
Strg+J	Popup-Menü für Vorlagen.
STRG+K+C	Fügt die Kommentarzeichen // vor jeder Zeile eines markierten Codeblocks hinzu oder entfernt diese.
Strg+K+E	Konvertiert das markierte Wort in Kleinschreibung.

Strg+K+F	Konvertiert das markierte Wort in Großschreibung.
Strg+L	Suchen Weitersuchen
Strg+P	Interpretiert das nächste Zeichen als Folge von ASCII-Code.
Strg+Q+[Sucht passenden Begrenzer (vorwärts).
Strg+Q+]	Sucht passenden Begrenzer (rückwärts).
Strg+Q+Strg+[Sucht passenden Begrenzer (vorwärts).
Strg+Q+Strg+]	Sucht passenden Begrenzer (rückwärts).
Strg+Q+Y	Löscht die Zeichen bis zum Ende der Zeile.
Strg+Umschalt+C	Aktiviert die Klassenvervollständigung für Klassendeklarationen, in welchen sich der Cursor befindet.
Strg+Umschalt+Ende	Markiert den Text zwischen Cursorposition und Dateiende.
Strg+Umschalt+Pos1	Markiert den Text zwischen Cursorposition und Dateianfang.
Strg+Umschalt+Links	Markiert das Wort links vom Cursor.
Strg+Umschalt+Bild Ab	Markiert den Text zwischen Cursorposition und unterem Bildschirmrand.
Strg+Umschalt+Bild Auf	Markiert den Text zwischen Cursorposition und oberem Bildschirmrand.
Strg+Umschalt+Rechts	Markiert das Wort rechts vom Cursor.
Strg+Umschalt+Tab	Zeigt das vorherige Fenster in der Pufferliste an.
Strg+T	Löscht das Wort links vom Cursor.
Strg+Tab	Zeigt das nächste Fenster in der Pufferliste an.
Strg+Y	Löscht die Zeichen bis zum Ende der Zeile.
Umschalt+Ab	Verschiebt den Cursor um einen Bildschirm nach UNTEN und markiert den übersprungenen Text.
Umschalt+Ende	Markiert Text von der Cursorposition bis zum Ende der aktuellen Zeile.
Umschalt+Eingabe	Fügt eine neue Zeile ein.
Umschalt+Pos1	Markiert Text von der Cursorposition bis zum Zeilenanfang.
Umschalt+Links	Markiert das Zeichen links vom Cursor.
Umschalt+Bild Ab	Verschiebt den Cursor um einen Bildschirm nach UNTEN und markiert den übersprungenen Text.
Umschalt+Bild Auf	Verschiebt den Cursor um einen Bildschirm nach oben und markiert den Text links von der Startposition des Cursors.
Umschalt+Rechts	Markiert das Zeichen rechts vom Cursor.
Umschalt+Leer	Fügt einen Leerzeichen ein.
Umschalt+Auf	Verschiebt den Cursor um eine Zeile nach oben und markiert den übersprungenen Text.

4.4.6 Tastaturvorlage Visual Basic-Emulation

Die folgende Tabelle enthält die Tastenkombinationen der Tastaturvorlage Visual Basic für den Quelltext-Editor.

Anmerkung: Wenn auf der Seite **Tools>Optionen>Editor-Optionen>Tastaturbelegung** die Option Strg+Alt-Tasten verwenden nicht markiert ist, sind die Tastenkombinationen, die Strg+Alt enthalten, deaktiviert.

Tastenkombination	Aktion
Alt + [Sucht passenden Begrenzer (vorwärts).
Alt +]	Sucht passenden Begrenzer (rückwärts).
Alt+F12	Symbol beim Cursor anzeigen (Delphi).
Alt+Bild Ab	Wechselt zur nächsten Registerkarte.
Alt+Bild Auf	Wechselt zur vorherigen Registerkarte.
Alt+Umschalt+Rück	Bearbeiten Widerrufen
Alt+Umschalt+Unten	Verschiebt den Cursor um eine Zeile nach unten und markiert die Spalte links von der Startposition des Cursors.
Alt+Umschalt+Ende	Markiert die Spalte von der Cursorposition bis zum Ende der aktuellen Zeile.
Alt+Umschalt+Pos1	Markiert die Spalte von der Cursorposition bis zum Zeilenanfang.
Alt+Umschalt+Links	Markiert die Spalte links vom Cursor.
Alt+Umschalt+Bild Ab	Verschiebt den Cursor um einen Bildschirm nach unten und markiert die Spalte links von der Startposition des Cursors.
Alt+Umschalt+Bild Auf	Verschiebt den Cursor um einen Bildschirm nach oben und markiert die Spalte links von der Startposition des Cursors.
Alt+Umschalt+Rechts	Markiert die Spalte rechts vom Cursor.
Alt+Umschalt+Oben	Verschiebt den Cursor um eine Zeile nach oben und markiert die Spalte links von der Startposition des Cursors.
Rück	Löscht das Zeichen links vom Cursor.
Strg+Alt+Umschalt+Ende	Markiert die Spalte von der Cursorposition bis zum Dateiende.
Strg+Alt+Umschalt+Pos1	Markiert die Spalte von der Cursorposition bis zum Dateianfang.
Strg+Alt+Umschalt+Links	Markiert die Spalte links vom Cursor.
Strg+Alt+Umschalt+Bild Ab	Markiert die Spalte von der Cursorposition bis zum Anfang des Bildschirms.
Strg+Alt+Umschalt+Bild Auf	Markiert die Spalte von der Cursorposition bis zum Ende des Bildschirms.
Strg+Alt+Umschalt+Rechts	Markiert die Spalte rechts vom Cursor.
Strg+Rück	Löscht das Wort links vom Cursor.
Strg+F4	Schließt die aktuelle Seite.
Strg+G	Öffnet eine Datei an der Cursorposition.
Strg+j	Popup-Menü für Vorlagen.
STRG+K+C	Fügt die Kommentarzeichen // vor jeder Zeile eines markierten Codeblocks hinzu oder entfernt diese.
Strg+K+E	Konvertiert das markierte Wort in Kleinschreibung.
Strg+K+F	Konvertiert das markierte Wort in Großschreibung.
Strg+L	Löscht eine Zeile.
Strg+P	Interpretiert das nächste Zeichen als Folge von ASCII-Code.
Strg+Q+[Sucht passenden Begrenzer (vorwärts).
Strg+Q+]	Sucht passenden Begrenzer (rückwärts).
Strg+Q+Strg+[Sucht passenden Begrenzer (vorwärts).

Strg+Q+Strg+]	Sucht passenden Begrenzer (rückwärts).
Strg+Q+Y	Löscht die Zeichen bis zum Ende der Zeile.
Strg+Umschalt+C	Aktiviert die Klassenvervollständigung für Klassendeklarationen, in welchen sich der Cursor befindet.
Strg+Umschalt+Ende	Markiert den Text zwischen Cursorposition und Dateiende.
Strg+Umschalt+Pos1	Markiert den Text zwischen Cursorposition und Dateianfang.
Strg+Umschalt+Links	Markiert das Wort links vom Cursor.
Strg+Umschalt+Bild Ab	Markiert den Text zwischen Cursorposition und unterem Bildschirmrand.
Strg+Umschalt+Bild Auf	Markiert den Text zwischen Cursorposition und oberem Bildschirmrand.
Strg+Umschalt+Rechts	Markiert das Wort rechts vom Cursor.
Strg+Umschalt+Tab	Zeigt das vorherige Fenster in der Pufferliste an.
Strg+T	Löscht das Wort links vom Cursor.
Strg+Tab	Zeigt das nächste Fenster in der Pufferliste an.
Strg+y	Löscht eine Zeile.
Strg+Y	Löscht die Zeichen bis zum Ende der Zeile.
Löschen	Löscht an der Cursorposition ein Zeichen oder einen Block.
Eingabe	Fügt eine neue Zeile ein.
Einfügen	Schaltet in den Einfügemodus um.
Umschalt+Rück	Löscht das Zeichen links vom Cursor.
Umschalt+Ab	Verschiebt den Cursor um einen Bildschirm nach UNTEN und markiert den übersprungenen Text.
Umschalt+Ende	Markiert Text von der Cursorposition bis zum Ende der aktuellen Zeile.
Umschalt+Eingabe	Fügt eine neue Zeile ein.
Umschalt+Pos1	Markiert Text von der Cursorposition bis zum Zeilenanfang.
Umschalt+Links	Markiert das Zeichen links vom Cursor.
Umschalt+Bild Ab	Verschiebt den Cursor um einen Bildschirm nach UNTEN und markiert den übersprungenen Text.
Umschalt+Bild Auf	Verschiebt den Cursor um einen Bildschirm nach oben und markiert den Text links von der Startposition des Cursors.
Umschalt+Rechts	Markiert das Zeichen rechts vom Cursor.
Umschalt+Leer	Fügt einen Leerzeichen ein.
Umschalt+Auf	Verschiebt den Cursor um eine Zeile nach oben und markiert den übersprungenen Text.
Tabulator	Fügt ein Tab-Zeichen ein.

5 Quelltexthaltepunkt hinzufügen/Adresshaltepunkt hinzufügen/Datenhaltepunkt hinzufügen

Start▸Haltepunkt hinzufügen

Verwenden Sie dieses Dialogfeld zum Festlegen eines Haltepunktes in einer Zeile Ihres Quelltextes, an einer Adresse oder bei Daten und zum Ändern der Eigenschaften eines vorhandenen Haltepunktes. Je nachdem, von wo aus das Dialogfeld aufgerufen wird, kann es auch den Titel Eigenschaft des Haltepunkts oder Eigenschaften des Adresshaltepunkts haben.

Element	Beschreibung
Dateiname (nur bei Quelltexthaltepunkten)	Gibt die Quelltextdatei für den Quelltexthaltepunkt an. Geben Sie den Namen einer Quelltextdatei für den Haltepunkt ein.
Zeilennummer (nur bei Quelltexthaltepunkten)	Setzt oder ändert die Zeilennummer für den Haltepunkt. Geben Sie die Zeilennummer für den Haltepunkt ein oder ändern Sie diese.
Adresse (nur bei Adress- und Datenhaltepunkten)	Gibt die Adresse für den Adresshaltepunkt an. Wenn die Adresse ausgeführt wird, hält die Programmausführung in der durch Bedingung und Durchlaufzähler bestimmten Weise an. Kann die Adresse mit einer Quelltextzeilennummer korreliert werden, so wird der Adresshaltepunkt als Quelltexthaltepunkt angelegt.
Länge (nur bei Datenhaltepunkten)	Legt die Länge des Datenhaltepunktes, beginnend bei "Adresse", fest. Die Länge wird für Standarddatentypen automatisch berechnet.
Bedingung	Legt einen Bedingungsausdruck fest, der bei jedem Durchlauf ausgewertet wird. Die Programmausführung wird angehalten, wenn der Ausdruck true ergibt. Geben Sie einen Bedingungsausdruck zum Beenden der Programmausführung ein. Sie können jeden gültigen Sprachausdruck verwenden. Alle Symbole im Ausdruck müssen jedoch vom Haltepunkt aus erreichbar sein. Funktionen sind zulässig, wenn sie einen Booleschen Wert zurückgeben. Falls bei Datenhaltepunkten keine Bedingung festgelegt ist, wird der Haltepunkt wirksam, wenn die Daten innerhalb des Bereichs, der im Feld Länge angegeben ist, geändert werden.

Durchlaufzähler	<p>Hält die Programmausführung an einer bestimmten Zeilennummer nach einer gegebenen Anzahl von Durchläufen an.</p> <p>Geben Sie die Anzahl der Durchläufe an. Der Debugger inkrementiert die Durchlaufzählung jedes Mal, wenn der Haltepunkt erreicht wird. Wenn der Durchlaufzähler die festgelegte Zahl erreicht hat, hält der Debugger die Programmausführung an. Wenn Sie beispielsweise 3 Durchläufe angeben, werden im Zähler die Werte 0 von 3, 1 von 3, 2 von 3 und 3 von 3 angezeigt. Die Programmausführung wird bei 3 von 3 angehalten.</p> <p>Da der Debugger den Zähler bei jedem Durchlauf inkrementiert, können Sie diese verwenden, festzustellen, welcher Schleifendurchlauf fehlschlägt. Setzen Sie den Durchlaufzähler auf die maximale Anzahl von Schleifendurchläufen und starten Sie Ihr Programm. Schlägt das Programm fehl, können Sie die Anzahl der Schleifendurchläufe berechnen, indem Sie die Anzahl der ausgeführten Durchläufe prüfen.</p> <p>Wenn Sie Durchlaufzähler mit Bedingungen verwenden, pausiert die Programmausführung, wenn der Bedingungsausdruck zum <i>n</i>-ten Mal true ergibt. Der Debugger dekrementiert den Durchlaufzähler nur dann, wenn der Bedingungsausdruck true ist.</p>
Gruppe	<p>Erstellt eine Haltepunktgruppe und fügt den Haltepunkt in diese Gruppe ein.</p> <p>Um eine bereits vorhandene Gruppe zu verwenden, wählen Sie sie aus der Dropdown-Liste aus. Haltepunktgruppen ermöglichen das Ausführen derselben Aktionen für alle Haltepunkte in der Gruppe.</p>
Weitere	Erweitert das Dialogfeld mit Feldern zum Zuordnen von Aktionen zu Haltepunkten
Anhalten	Hält die Ausführung an. Dies ist die herkömmliche Standardaktion eines Haltepunkts.
Spätere ignorieren	<p>Exceptions</p> <p>Ignoriert alle nachfolgenden Exceptions, die vom aktiven Prozess in der aktuellen Debugging-Sitzung ausgelöst werden (der Debugger hält nicht bei jeder Exception an). Verwenden Sie diese und die Option Spätere Exceptions behandeln paarweise. Sie können bestimmte Anweisungsblöcke in ein <i>Ignorieren / Behandeln</i>-Paar einschließen und dadurch verhindern, dass der Debugger bei Exceptions in diesem Quelltext anhält.</p>
Spätere behandeln	<p>Exceptions</p> <p>Behandelt alle nachfolgenden Exceptions, die vom aktiven Prozess in der aktuellen Debugging-Sitzung ausgelöst werden (der Debugger hält dann entsprechend der Einstellungen in Tools Optionen Debugger-Optionen Sprach-Exceptions an). Diese Option hält bei allen Exceptions an. Mit ihr können Sie das normale Verhalten wiederherstellen, wenn die Exception-Behandlung zuvor mit Spätere Exceptions ignorieren ausgeschaltet wurde.</p>
Meldung protokollieren	Schreibt die angegebene Meldung in das Ereignisprotokoll. Sie können den Wortlaut dieser Meldung eingeben.
Eval-Ausdruck	Wertet den angegebenen Ausdruck aus und schreibt - da die Option Ergebnis protokollieren standardmäßig aktiviert ist - das Ergebnis der Auswertung in das Ereignisprotokoll. Deaktivieren Sie Ergebnis protokollieren, wenn die Auswertung ohne Protokollierung durchgeführt werden soll.
Ergebnis protokollieren	Schreibt das Ergebnis der Auswertung in das Ereignisprotokoll, wenn Text in das Feld Eval-Ausdruck eingegeben wird. Bei deaktivierter Option findet keine Protokollierung statt.
Gruppe aktivieren	Aktiviert alle Haltepunkte der angegebenen Gruppe. Wählen Sie den gewünschten Gruppennamen aus.
Gruppe deaktivieren	Deaktiviert alle Haltepunkte der angegebenen Gruppe. Wählen Sie den gewünschten Gruppennamen aus.
Aufruf-Stack protokollieren	<p>Zeigt den gesamten oder einen Teil des Aufruf-Stack im Fenster Ereignisprotokoll an, wenn ein Haltepunkt erreicht wird.</p> <p>Gesamter Stack zeigt den gesamten Aufruf-Stack an.</p> <p>Teil-Stack zeigt nur die Anzahl der im Feld Anzahl der Frames festgelegten Frames an.</p>

Siehe auch

Quelltexthaltepunkte setzen und bearbeiten ( siehe Seite 19)

6 Compilieren, Build erstellen und Anwendungen ausführen

Während der Entwicklung einer Anwendung in der IDE können Sie diese jederzeit compilieren, ein Build erstellen und die Anwendung in der IDE ausführen. Jede dieser Operationen kann eine ausführbare Datei (wie z.B. `.exe`, `.dll`, `.obj` oder `.bpl`) produzieren. Die drei Operationen unterscheiden sich jedoch in ihrem Verhalten:

- **Compilieren (Projekt>Compilieren):** Compiliert nur die Dateien, die seit dem letzten Build geändert wurden und die davon abhängigen Dateien. Compilieren für die Anwendung nicht aus (siehe **Start**).
- **Erzeugen (Projekt>Erzeugen):** Wenn Sie ein Projekt erstellen, wird der gesamte Quelltext des aktuellen Projekts, unabhängig davon, ob sich dieser geändert hat, neu compiliert. Dies ist dann sinnvoll, wenn nicht mehr nachvollziehbar ist, welche Dateien sich geändert haben oder wenn die Projekt- oder Compiler-Optionen neu eingestellt wurden.
- **Start (Start>Start (F9):** Bei der Ausführung eines Projekts wird nur der geänderte Quelltext compiliert und die Anwendung, sofern die Compilierung erfolgreich war, auch ausgeführt. Anschließend kann die Anwendung in der IDE verwendet und getestet werden.

Mit dem Kontextmenübefehl Bereinigen des Projektknotens in der Projektverwaltung löschen Sie alle erzeugten Dateien eines vorherigen Builds.

Compiler-Optionen

Für ein Projekt lassen sich viele Compiler-Optionen einstellen, indem Sie **Projekt>Optionen** wählen und dann die Compiler-bezogenen Seiten auswählen. Die meisten Optionen auf den Seiten Delphi-Compiler und C++-Compiler entsprechen den Compiler-Optionen, die in der Online-Hilfe für die jeweilige Seite beschrieben sind.

Um zusätzliche Compiler-Optionen anzugeben, können Sie den Compiler von der Befehlszeile aus aufrufen. Eine vollständige Liste aller Optionen des Delphi-Compilers sowie Informationen zum Ausführen des Delphi-Compilers in der Befehlszeile finden Sie im **Inhalt der Delphi-Sprachreferenz**.

Compiler-Status und -Informationen

Wenn bei der Compilierung eines Projekts die Compiler-Optionen im Fenster Meldungen angezeigt werden sollen, wählen Sie **Tools>Optionen>Umgebungsoptionen** und aktivieren die Option Befehlszeile anzeigen. Beim nächsten Compilieren eines Projekts werden der Befehl, der zur Compilierung des Projekts verwendet wurde, und die Antwortdatei im Fenster Meldungen angezeigt. In der Antwortdatei sind die Compiler-Optionen und die zu compilierenden Dateien aufgeführt.

Nachdem ein Projekt compiliert wurde, können Sie Informationen darüber anzeigen, indem Sie **Projekt>Infos** wählen. Das

Dialogfeld Information zeigt die Anzahl der compilierten Quelltextzeilen, die Bytegröße des Quelltextes und der Daten, die Stack- und Dateigröße und den Compilier-Status des Projekts an.

Compiler-Fehler

Wenn Sie ein Projekt compilieren, werden die Compiler-Meldungen im Fenster Meldungen angezeigt. Um Erläuterungen zu einer Meldung anzuzeigen, markieren Sie diese und drücken die Taste F1.

Build-Konfigurationen

Sie können bestimmte Projektoptionen als eine benannte Build-Konfiguration speichern. Für jedes Projekt lässt sich eine aktive Build-Konfiguration festlegen, die Sie während der Projektentwicklung ändern können. Sie können beispielsweise speziell für das Debuggen Ihres Projekts Projektoptionen setzen, diese Optionen als **Debug**-Build-Konfiguration speichern und diese dann beim Debuggen des Projekts als aktive Build-Konfiguration übernehmen.

Build-Konfigurationen bestehen aus Optionen, die auf den Registerkarten Compiler, Compiler-Meldungen, Linker und Verzeichnisse/Bedingungen des Dialogfeldes **Projekt > Optionen** festgelegt werden.

Standardmäßig enthält die IDE eine Debug-Konfiguration und eine Release-Konfiguration. Sie können die Optionen dieser Standardkonfigurationen ändern oder neue, eigene Build-Konfigurationen erstellen. Mit dem Manager für Build-Konfigurationen können Sie eine ausgewählte Build-Konfiguration für bestimmte Projekte oder Projektgruppen übernehmen.

Ein Projekt mit MSBuild erzeugen

Wenn Sie ein Projekt explizit erzeugen, ruft die IDE MSBuild, die Microsoft Build-Engine, auf. Der Build-Prozess ist für Entwickler vollkommen transparent. MSBuild wird als Teil der Befehle Compilieren, Erzeugen und Start aufgerufen, die in den Menüs Projekt bzw. Start verfügbar sind.. MSBuild.exe können Sie auch von der Befehlszeile aus oder über die MSBuild-Konsole, die im Menü **Start** zur Verfügung steht, aufrufen.

Siehe auch

Überblick zu MSBuild ([siehe Seite 1461](#))

Überblick zu Build-Konfigurationen ([siehe Seite 1463](#))

Packages erstellen ([siehe Seite 3](#))

Seite Compiler (Online-Hilfe) ([siehe Seite 501](#))

Manager für Build-Konfigurationen ([siehe Seite 496](#))

Erstellen von benannten Build-Konfigurationen (C++) ([siehe Seite 6](#))

Erstellen von benannten Build-Konfigurationen (Delphi) ([siehe Seite 7](#))

Erstellen eines Projekts von der Befehlszeile aus ([siehe Seite 12](#))

7 Einführung

Die integrierte Entwicklungsumgebung (IDE) von RAD Studio enthält viele Tools und Funktionen zur schnellen Entwicklung leistungstarker .NET-Anwendungen. Einige Funktionen und Tools stehen in bestimmten Editionen von RAD Studio nicht zur Verfügung. Eine Liste der Funktionen und Tools, die in Ihrer Edition verfügbar sind, finden Sie in der Funktionsmatrix unter <http://www.codegear.com>.

7.1 Was ist RAD Studio?

RAD Studio ist eine integrierte Entwicklungsumgebung (IDE) für das Erstellen von Delphi Win32-Anwendungen. Die IDE von RAD Studio enthält eine umfangreiche Sammlung von Tools, die Sie dabei unterstützen, den Entwicklungszyklus möglichst effizient und einfach zu gestalten. Die Tools, die in der IDE angeboten werden, variieren je nach RAD Studio-Version. In den nachstehenden Abschnitten werden diese Tools kurz erläutert.

Anwendungen modellieren

Durch Modellieren können Sie die Performance, Effektivität und Verwaltbarkeit Ihrer Anwendungen verbessern, da Sie bereits vor dem Verfassen der ersten Codezeile über ein detailliertes visuelles Design verfügen. RAD Studio stellt das Modellierungstool **Together** zur Verfügung, das in der IDE ausgeführt wird.

Benutzeroberflächen entwerfen

Mit den visuellen Designern von RAD Studio können Sie grafische Benutzeroberflächen erstellen, indem Sie die erforderlichen Komponenten mit der Maus aus der Tool-Palette in ein Formular ziehen. Mithilfe der Designer lassen sich Windows Forms-Anwendungen erstellen, die die umfangreiche VCL (Visual Component Library) verwenden. Sie können Ihre Anwendungen auch für verschiedene Versionen von Windows, einschließlich Windows Vista, anpassen.

Quelltext erzeugen und bearbeiten

RAD Studio erzeugt den größten Teil des Anwendungscodes automatisch, wenn Sie mit einem neuen Projekt beginnen. Für eine zügige Fertigstellung der restlichen Anwendungslogik stellt Ihnen der textbasierte Quelltext-Editor zahlreiche Funktionen zur Verfügung, beispielsweise Refactoring, synchronisierte Bearbeitung, Programmierhilfe, Aufzeichnung von Tastatur-Makros und benutzerdefinierte Tastaturbelegungen. Die Syntaxhervorhebung und das Code-Folding ermöglichen ein einfaches Lesen und Navigieren im Quelltext.

Compilierung, Debugging und Deployment von Anwendungen

In der IDE lassen sich Compiler-Optionen einstellen, Anwendungen ausführen und Compiler-Meldungen anzeigen. In RAD Studio ist die Build-Engine MSBuild integriert. Die Befehle Compilieren und Erzeugen rufen MSBuild auf. MSBuild kann auch über die Befehlszeile oder dem Startmenübefehl **RAD Studio-Befehlszeile** aufgerufen werden. Der Befehl **RAD Studio-Befehlszeile** öffnet ein Befehlskonsolenfenster und setzt automatisch den Pfad auf die ausführbare Datei von MSBuild und die Umgebungsvariable BDS auf Ihr Installationsverzeichnis.

Compiler-Optionen und mehrere andere **Projekt>Optionen** lassen sich als benannte Build-Konfigurationen speichern und können für bestimmte Projekte übernommen werden. Standardmäßig enthält die IDE eine **Debug**- und eine **Ausgabe**-Build-Konfiguration.

Der integrierte Win32-Debugger unterstützt Sie bei der Behebung von Laufzeit- oder Logikfehlern, bei der Prüfung der Programmausführung und bei der Beobachtung von Variablen sowie der Bearbeitung von Datenwerten. RAD Studio enthält InstallAware zum Erstellen von Windows Installations-Setups.

Siehe auch

Benutzeroberflächen entwerfen ([siehe Seite 1410](#))

Quelltext-Editor ([siehe Seite 1373](#))

Compilieren ([siehe Seite 1351](#))

Anwendungen debuggen ([siehe Seite 1439](#))

Deployment von Anwendungen ([siehe Seite 1443](#))

7.2 Neuerungen in RAD Studio (Delphi)

RAD Studio stellt neue Leistungsmerkmale zur Entwicklung von Delphi-Anwendungen für Win32 bereit.

IDE

- **MSBuild:** Die IDE unterstützt jetzt die MSBuild-Build-Engine anstelle des früheren internen Build-Systems. Wenn Sie ein bereits vorhandenes Projekt öffnen, konvertiert die IDE das Projekt automatisch in das MSBuild-Format und ändert die Projekterweiterung. Sie können auch die MSBuild-Konsole (über das Startmenü) oder die Datei MSBuild.exe verwenden, um Projekte von der Befehlszeile aus zu erzeugen.
- **Build-Ereignisse:** Sie können DOS-Befehle und Makros angeben, die nach oder vor dem Compilieren Ihres Projekts ausgeführt werden sollen.
- **Build-Konfigurationen:** Sie können jetzt im Fenster Projektoptionen benannte Build-Konfigurationen erstellen. Um einem Projekt oder einer Projektgruppe eine benannte Build-Konfiguration zuzuweisen, verwenden Sie den neuen Manager für Build-Konfigurationen, der über das Menü Projekt aufgerufen wird.
- **Vista- und XP-Themes:** Die IDE unterstützt nun Vista- und XP-Themes. Themes sind standardmäßig aktiviert, Sie können sie aber für die IDE oder einzelne Anwendungen deaktivieren.
- **Mehrfachauswahl in der Projektverwaltung:** In der Projektverwaltung können Sie für die Kontextmenübefehle Öffnen, Speichern, Speichern unter und Aus dem Projekt entfernen mehrere Dateien auswählen.
- **Neuer Datei-Browser:** Sie können den neuen Datei-Browser zu Anzeigen von Dateien auf der Festplatte und zum Interagieren mit der Windows-Shell verwenden.

Debugger

- **Verschieben des Ausschnitts des Ereignisprotokolls verhindern:** Eine neue Option auf der Seite [Tools>Optionen>Debugger-Optionen>Ereignisprotokoll](#) verhindert, dass der Ausschnitt des Ereignisprotokolls beim Auftreten neuer Ereignisse so verschoben wird, dass diese Ereignisse eingeblendet werden.
- **CPU-Fenster:** Sie können jetzt einzelne Bereiche des CPU-Fensters, wie z.B. die Bereiche Disassembly, CPU-Stack und Register, öffnen. Diese einzelnen Bereiche des CPU-Fensters sind andockbar; Sie können die Bereiche abdocken und an einer anderen Position in der IDE wieder andocken. Das CPU-Fenster wird nun auch automatisch geschlossen, wenn Sie eine Debug-Sitzung beenden. Der Bereich Disassembly enthält zwei neue Optionen (Opcodes anzeigen und Adressen anzeigen).
- **Fenster: Aufruf-Stack:** Sie können nun einen Haltepunkt für einen bestimmten Frame setzen.
- **Nicht-Benutzer-Haltepunkte ignorieren:** Sie können nun festlegen, dass der Debugger Haltepunkte ignorieren soll, die Sie nicht extra in der IDE gesetzt haben.
- **Debug-Quellpfad:** Der Quellpfad für das Debuggen ist nun eine globale Einstellung, die Sie auf der Registerkarte [Projekt>Optionen>Debugger](#) festlegen.
- **Neue Schaltflächen in der Symbolleiste:** Der Befehl Bei Sprach-Exceptions benachrichtigen steht nun als Symbol ([Ansicht>Symbolleisten>Anpassen>Anweisungen>Kategorien>Start](#)) zur Verfügung. Für einen schnellen Zugriff darauf, können Sie das Symbol auf die Symbolleiste ziehen.
- **Transparente Kurzhinweise:** Um einen Kurzhinweis des Debugger-Evaluators transparent anzuzeigen, drücken Sie die Taste STRG, wenn der Kurzhinweis angezeigt wird. Bei transparenten Kurzhinweisen lassen sich die dahinter liegenden Elemente einsehen.

Datenbank

Es wurden viele Änderungen vorgenommen, um die Datenbankentwicklung in RAD Studio besser zu unterstützen.

dbExpress

Unicode-Unterstützung wurde zu den Oracle-, Interbase- und MySQL dbExpress-Treibern hinzugefügt.

Neue Treiber-Clients wurden hinzugefügt: Interbase 2007 und MySQL 4.1 und 5.

Ein neues dbExpress-Framework wurde erstellt. Sie können mit diesem Framework an vorhandene Treibern ankoppeln oder neue Treiber durch Erweiterung der abstrakten Klassen des dbExpress-Framework schreiben. Dieses Framework lässt sich direkt für native und verwaltete Anwendungen einsetzen.

Ein Delegattreiber ist ein Treiber zwischen der Anwendung und dem eigentlichen Treiber. Delegattreiber ermöglichen eine Vor- und Nachverarbeitung aller public Methoden und Eigenschaften des dbExpress 4-Framework. Delegattreiber sind beim Verbindungs-Pooling, beim Erstellen von Treiberprofilen, bei der Ablaufverfolgung und beim Auditing hilfreich. Beispiele für Delegattreiber sind im Produkt enthalten.

Die API von dbExpress VCL-Komponenten wurde leicht verändert. Die meisten Anwendungen sind von den Änderungen der dbExpress VCL nicht betroffen. Es gibt aber einige Methoden, Eigenschaften, Ereignisse, Konstanten und Aufzählungen, die entfernt oder durch äquivalente Funktionen ersetzt wurden.

Auch die dbExpress VCL-Komponenten, die in der obersten Schicht des Framework angeordnet sind, können für native und verwaltete Anwendungen verwendet werden.. In den VCL-Komponenten wurden geringe API-Änderungen an der Klasse TSQLConnection (Methodenänderungen), an der Klasse TSQLDataSet (neue Eigenschaft) und an Datenstrukturen (einige wurden entfernt oder ersetzt) vorgenommen. Unter dbExpress Framework-Kompatibilität finden Sie nähere Informationen.

Das dbExpress-Treiber-Framework:

- ist vollständig in der Sprache Delphi geschrieben und ermöglicht das Schreiben von Treibern in Delphi.
- verwendet einen streng typisierten Datenzugriff anstelle von Zeigern. Das Framework verwendet z.B. String-Typen anstatt Zeiger auf Strings.
- verwendet einen einzigen Quelltext. Das bedeutet, dass eine einzige Kopie des Quelltextes entweder mit den nativen DCC32- oder den verwalteten DCCIL-Compilern compiliert werden kann.
- hat nur abstrakte Basisklassen, die für Treiber, Verbindungen, Befehle, Reader usw. verwendet werden.
- verwendet eine auf Exception basierte Fehlerbehandlung anstelle der Rückgabe von Fehlercodes.

VCL

AJAX: RAD Studio unterstützt die AJAX-basierte RAD VCL für die Web-Entwicklung.

Microsoft Vista-Kompatibilität: RAD Studio stellt Komponenten, Klassen, Methoden und Eigenschaften bereit, die mit dem Erscheinungsbild des Vista-Betriebssystems kompatibel sind.

Neue Komponenten: Die VCL (Visual Component Library) unterstützt die folgenden neuen Komponenten:

- TFileOpenDialog
- TFileSaveDialog
- TTaskDialog

Neue Klassen: Die folgenden Klassen wurden hinzugefügt:

- TCustomFileDialog
- TCustomFileOpenDialog
- TCustomFileSaveDialog
- TCustomTaskDialog
- TFavoriteLinkItem
- TFavoriteLinkItems
- TFavoriteLinkItemsEnumerator
- TFileTypeItem
- TFileTypeItems

- `TTaskDialogBaseButtonItem`
- `TTaskDialogButtonItem`
- `TTaskDialogButtons`
- `TTaskDialogButtonsEnumerator`
- `TTaskDialogProgressBar`
- `TTaskDialogRadioButtonItem`

Siehe auch

Überblick zu MSBuild ([siehe Seite 1461](#))

Erstellen von Build-Ereignissen ([siehe Seite 5](#))

Manager für Build-Konfigurationen ([siehe Seite 496](#))

Deaktivieren von Themes für die IDE und für die Anwendung ([siehe Seite 65](#))

dbExpress Framework-Kompatibilität

Datei-Browser ([siehe Seite 806](#))

7.3 Neuerungen in RAD Studio (C++Builder)

RAD Studio stellt neue Leistungsmerkmale zur Entwicklung von C++-Anwendungen bereit.

C++Builder 2007

Die folgenden Features sind neu oder wurden signifikant verändert:

- **MSBuild ist die neue Build-Engine:** Beim Erzeugen eines C++-Projekts übernimmt MSBuild jetzt die Ausführung des Build-Prozesses. Die Struktur der Projektdatei wurde in XML geändert und enthält nun die für MSBuild erforderlichen Optionen. Die Erweiterung der Projektdatei wurde in `.cbproj` geändert. Sie können Projekte mit der MSBuild-Befehlssyntax von der Befehlszeile aus erzeugen. Weitere Informationen finden Sie unter Überblick zu MSBuild.
- **Das Dialogfeld Projekt>Optionen wurde umgestaltet: Es wurden neue Seiten hinzugefügt, und einige vorhandene Seiten wurden aus Gründen einer besseren Organisation der Optionen umbenannt. Neue Optionen, wie z.B. `-vb`-Optionen zur Unterstützung von C++-Konstrukten, die standardmäßig nicht mehr unterstützt werden, wurden ebenfalls hinzugefügt. Sie können mit der neuen Seite Projekteigenschaften festlegen, dass der C++-Compiler die Bibliothekspfade verwalten, Package-Importe überprüfen, Header-Abhängigkeiten anzeigen oder die Auto-Abhängigkeitsprüfung verwenden soll. Weitere Informationen finden Sie unter Projektoptionen einstellen (siehe Seite 75). Informationen über Projektoptionen, die in der IDE nicht mehr verfügbar sind, finden Sie unter Nicht verfügbare Optionen (siehe Seite 587).**
- **Sie können Projektoptionen zusammenführen:** Einige Projektoptionen verfügen über das Kontrollfeld Zusammenführen. Wird dieses Feld markiert, bezieht die IDE die Optionswerte des unmittelbaren Vorfahren der aktuellen Build-Konfiguration ein. Die Optionen der aktuellen Konfiguration werden dadurch nicht geändert. Weitere Informationen zum Zusammenführen finden Sie unter Projektoptionen (siehe Seite 517).
- **Build-Konfigurationen sind nun umfassender:** Build-Konfigurationen wurden geändert. Eine Build-Konfiguration enthält die Optionen, die Sie auf vielen Seiten des Dialogfeldes **Projekt>Optionen** festgelegt haben. Build-Konfigurationen speichern Befehlszeilenoptionen für Build-Tools, wie den Compiler, den Linker und MSBuild. Sie können eigene Konfigurationen erstellen, aber es gibt auch drei Standardkonfigurationen (Base, Debug und Ausgabe). Weitere Informationen finden Sie unter Überblick über Build-Konfigurationen (C++) (siehe Seite 1465).
- **Benannte Optionsgruppen sind neu:** Sie können auf den Build-bezogenen Seiten des Dialogfeldes **Projekt>Optionen** benannte Optionsgruppen erstellen und übernehmen. Benannte Optionsgruppen werden in Dateien mit der Erweiterung `.optset` gespeichert. Weitere Informationen finden Sie unter Überblick über benannte Optionsgruppen (siehe Seite 1469).
- **Die Build-Reihenfolge wurde geändert:** MSBuild erzeugt Dateien entsprechend dem Typ (Erweiterung) und nicht anhand der vom Benutzer änderbaren Reihenfolge, die früher verwendet wurde. Die neue Build-Reihenfolge ist Delphi (`.pas`), C/C++ (`.c/.cpp`), Assembler (`.asm`), dann Ressource (`.rc`). In jedem Ordner oder virtuellen Ordner werden die Dateien entsprechend ihres Dateityps erzeugt. Weitere Informationen finden Sie unter Überblick über virtuelle Ordner.
- **Der neue Manager für Build-Konfigurationen aktiviert eine Build-Konfiguration:** Wählen Sie **Projekt>Konfigurations-Manager**, um die Build-Konfiguration festzulegen, die für das ausgewählte Projekt (oder die Projekte) aktiv sein soll. Der Konfigurations-Manager ersetzt die alte Art des Festlegens der aktiven Konfiguration für C++-Projekte. Weitere Informationen finden Sie unter Manager für Build-Konfigurationen (siehe Seite 496).
- **Sie können Build-Ereignisse festlegen:** Sie können Befehle angeben, die an bestimmten Punkten des Build-Prozesses ausgeführt werden sollen (Pre-Link-Ereignisse sind neu; Pre-Build- und Post-Build-Ereignisse gab es schon in früheren Versionen). Klicken Sie in der Projektverwaltung mit der rechten Maustaste auf eine Build-fähige Datei, und wählen Sie Build-Ereignisse. Weitere Informationen finden Sie unter Build-Ereignisse (Dialogfeld) (siehe Seite 497).
- **Sie können .targets-Dateien erstellen und einem Projekt hinzufügen:** Eine `.targets`-Datei ist eine XML-Datei, die MSBuild-Scripts, wie z.B. Listen von auszuführenden Tasks, enthalten. Weitere Informationen finden Sie unter Targets-Dateien (siehe Seite 1471).
- **Der Speicherort des Democodes wurde geändert:** Der Democode befindet sich jetzt in **Eigene Dateien\RAD Studio\Demos**. Die Demos wurden aus dem Verzeichnis **Programme** wegen der Beschränkungen von Microsoft Vista verlagert.

- **Sie können C++-Packages mit Delphi compilieren:** C++Builder unterstützt das Compilieren von Entwurfszeit-Packages, die Delphi-Quelltextdateien enthalten. Falls jedoch eine Delphi-Quelltextdatei einen Verweis auf von der IDE bereitgestellte Entwurfszeit-Units, wie z.B. DesignIntf, DesignEditors oder ToolsAPI aus der Datei DesignIDE100.bpl enthält, müssen Sie Vorkehrungen treffen, damit die Verweise vom C++Builder-Package aufgelöst werden können. Unter Compilieren von Entwurfszeit-Packages (siehe Seite 4) halten Sie eine Anleitung dazu.

Testen von Units für C++

Über das DUnit Testing Framework wurde die Unterstützung für das Testen von Units integriert. Das DUnit-Framework basiert auf dem JUnit-Test-Framework und enthält größtenteils dieselbe Funktionalität.

Mit den Experten zum Testen von Units in C++Builder erzeugen Sie schnell Skeleton-Templates für das Testprojekt, erstellen und löschen Methoden, und Basistests. Anschließend können die Templates verändert werden, indem Sie eine spezielle Testlogik zum Testen von bestimmten Methoden hinzufügen.

Tests können entweder in der Konsolen-Test-Runner oder im DUnit GUI Test-Runner ausgeführt werden. Der Konsolen-Test-Runner leitet die Testausgabe an die Konsole. Der DUnit GUI Test-Runner zeigt die Testergebnisse interaktiv in einem GUI-Fenster mit Farbcodierungen zum Kennzeichnen von Erfolg oder Fehlschlag an.

Erweiterung der Unterstützung von Web-Services in C++Builder

Die Unterstützung von Web-Services in C++Builder umfasst nun das Folgende:

- Nicht gebundene Elemente
- Optionale Elemente
- "Nullable" Elemente
- WSDL und Schema, das externe Schemas importiert

Diese Erweiterungen stellen die Unterstützung von Web-Services in C++Builder auf eine Stufe mit der in Delphi und ermöglichen Ihren Anwendungen die Interaktion mit robusteren Web-Services, wie eBay, Amazon, MapPoint usw.

IDE

Die folgenden Features sind in der IDE (integrierte Entwicklungsumgebung) neu oder wurden signifikant verändert:

- **Vista- und XP-Themes:** Die IDE unterstützt nun Windows Vista- und XP-Themes. Themes sind standardmäßig aktiviert, Sie können sie aber für die IDE oder einzelne Anwendungen deaktivieren. Weitere Informationen finden Sie unter IDE unter Windows Vista.
- **Doppelte Dateinamen:** Ein Projekt darf jetzt mehrere Dateien mit demselben Namen enthalten. Beispielsweise kann Common\source1.cpp und Product\source1.cpp in einem Projekt vorhanden sein. Die IDE verwaltet die Erzeugung von Objektdateien so, dass es keine Konflikte gibt und dass das Objekt beider Dateien beim Build-Prozess des Projekts verwendet wird.
- **Ausführlichere Hilfe zum Speichermanager:** Der neue mit Borland Developer Studio 2006 herausgegebene Speichermanager ist in diesem Release von RAD Studio vollständig dokumentiert. Folgende Themen sind enthalten: Konfigurieren des Speichermanagers, Überwachen des Speichermanagers und Gemeinsame Nutzung des Speichermanagers mit ShareMem und SimpleShareMem. Die Routinen und Variablen des Speichermanagers sind in diesem Thema unter VCL aufgeführt. Weitere Informationen finden Sie unter Überblick zur Speicherverwaltung (siehe Seite 1017).
- **Mehrfachauswahl in der Projektverwaltung:** Halten Sie die Taste STRG gedrückt, um in der Projektverwaltung für die Kontextmenübefehle Öffnen, Speichern, Speichern unter und Aus dem Projekt entfernen mehrere Dateien auszuwählen.
- **Neuer Datei-Browser:** Mit **Ansicht>Datei-Browser** öffnen Sie den Datei-Browser, um grundlegende Dateioperationen auszuführen oder den SVN-Status einer Datei anzuzeigen. Weitere Informationen finden Sie unter Datei-Browser (siehe Seite 806).
- **Neues Symbolleiste in der Strukturansicht:** Mit Hilfe einer neuen, nur für C++ verfügbaren Symbolleiste können Sie die folgenden Funktionen ausführen: Alphabetisch sortieren, Nach Typ gruppieren, Nach Sichtbarkeit sortieren, Typ anzeigen und Sichtbarkeit anzeigen. Weitere Informationen finden Sie unter Ansicht>Struktur.
- **Virtuelle Ordner in der Projektverwaltung:** Sie können virtuelle Verknüpfungen zwischen Elementen in der Baumstruktur erstellen. Mit virtuellen Ordnern kann die Build-Reihenfolge beeinflusst werden. Weitere Informationen finden Sie unter Überblick über virtuelle Ordner.

Debugger

Die folgenden Features sind neu oder wurden signifikant verändert:

- **Verschieben des Ausschnitts des Ereignisprotokolls verhindern:** Eine neue Option auf der Seite **Tools**▶**Optionen**▶**Debugger-Optionen**▶**Ereignisprotokoll** verhindert, dass der Ausschnitt des Ereignisprotokolls beim Auftreten neuer Ereignisse so verschoben wird, dass diese Ereignisse eingeblendet werden.
- **CPU-Fenster:** Sie können jetzt einzelne Bereiche des CPU-Fensters, wie z.B. die Bereiche Disassembly, CPU-Stack und Register, öffnen. Diese einzelnen Bereiche des CPU-Fensters sind andockbar; Sie können die Bereiche abdocken und an einer anderen Position in der IDE wieder andocken. Das CPU-Fenster wird nun auch automatisch geschlossen, wenn Sie eine Debug-Sitzung beenden. Der Bereich Disassembly enthält zwei neue Optionen (Opcodes anzeigen und Adressen anzeigen). Weitere Informationen finden Sie unter CPU-Fenster (siehe Seite 786).
- **Fenster: Aufruf-Stack:** Sie können nun einen Haltepunkt für einen bestimmten Frame setzen. Weitere Informationen finden Sie unter Aufruf-Stack-Fenster (siehe Seite 785).
- **Nicht-Benutzer-Haltepunkte ignorieren:** Sie können nun festlegen, dass der Debugger Haltepunkte ignorieren soll, die Sie nicht extra in der IDE gesetzt haben. Weitere Informationen finden Sie unter CodeGear-Debugger (siehe Seite 726).
- **Debug-Quellpfad:** Der Quellpfad für das Debuggen ist nun eine globale Einstellung, die Sie auf der Registerkarte **Projekt**▶**Optionen**▶**Debugger** festlegen. Weitere Informationen finden Sie unter Die Suchreihenfolge für Debug-Symboltabellen festlegen (siehe Seite 36).
- **Neue Schaltflächen in der Symbolleiste:** Der Befehl Bei Sprach-Exceptions benachrichtigen steht nun als Symbol (**Ansicht**▶**Symbolleisten**▶**Anpassen**▶**Anweisungen**▶**Kategorien**▶**Start**) zur Verfügung. Für einen schnellen Zugriff darauf, können Sie das Symbol auf die Symbolleiste ziehen. Weitere Informationen finden Sie unter Sprach-Exceptions (siehe Seite 753).
- **Transparente Kurzhinweise:** Um einen Kurzhinweis des Debugger-Evaluators transparent anzeigen, drücken Sie die Taste **STRG**, wenn der Kurzhinweis angezeigt wird. Bei transparenten Kurzhinweisen lassen sich die dahinter liegenden Elemente einsehen.

Together-Modellierung

Die Unterstützung für das Together-Modellierungstool ist neu in C++:

- C++Builder 2007 stellt eine begrenzte Modellierungsunterstützung für Together bereit, dem vollständig in der IDE integrierten Modellierungstool. Beachten Sie bitte, dass nur die Code-Visualisierung (schreibgeschützt), die Dokumentationserzeugung und die Funktionen zum Drucken eines Diagramms in C++Builder 2007 verfügbar sind, obwohl in der Online-Hilfe die gesamten Together-Features beschrieben sind.
- **C++-Klassendiagramme (Code-Visualisierung):** Das C++-Klassendiagramm steht nur im schreibgeschützten Modus zur Verfügung. Sie können in Ihren C++-Projekten Entwurfsdiagramm erstellen, aber Sie können in der Modellansicht keine Klassen, Interfaces usw. erstellen.
- **Entwurfsdiagramme:** Das komplette Set der Entwurfsdiagramme ist nur in der Enterprise-Edition des Produkts verfügbar. Dazu gehören Sequenzdiagramme, Kollaborationsdiagramme, Zustandsdiagramme, Verteilungsdiagramme, Anwendungsfalldiagramme, Aktivitätsdiagramme und Komponentendiagramme.
- **Drucken von Diagrammen und Erzeugen der Dokumentation:** Sowohl die Professional- als auch die Enterprise-Edition unterstützt das Drucken von Diagrammen. Die Enterprise-Edition unterstützt außerdem die Dokumentationserzeugung.

Weitere Informationen finden Sie unter Einführung in Together (siehe Seite 1380) oder [Modellieren von Anwendungen mit Together](#).

Anmerkung: Nur bestimmte Produkteditionen enthalten den gesamten Funktionsumfang, der in den Together-Themen dieses Hilfesystems beschrieben ist. Das aktuelle Release enthält ein begrenztes Funktionsset.

Datenbank

Es wurden viele Änderungen vorgenommen, um die Datenbankentwicklung in RAD Studio besser zu unterstützen.

dbExpress

Unicode-Unterstützung wurde zu den Oracle-, Interbase- und MySQL dbExpress-Treibern hinzugefügt.

Neue Treiber-Clients wurden hinzugefügt: Interbase 2007 und MySQL 4.1 und 5.

Ein neues dbExpress-Framework wurde erstellt. Sie können mit diesem Framework an vorhandene Treibern ankoppeln oder neue Treiber durch Erweiterung der abstrakten Klassen des dbExpress-Framework schreiben. Dieses Framework lässt sich direkt für native und verwaltete Anwendungen einsetzen.

Ein Delegattreiber ist ein Treiber zwischen der Anwendung und dem eigentlichen Treiber. Delegattreiber ermöglichen eine Vor- und Nachverarbeitung aller public Methoden und Eigenschaften des dbExpress 4-Framework. Delegattreiber sind beim Verbindungs-Pooling, beim Erstellen von Treiberprofilen, bei der Ablaufverfolgung und beim Auditing hilfreich. Beispiele für Delegattreiber sind im Produkt enthalten.

Die API von dbExpress VCL-Komponenten wurde leicht verändert. Die meisten Anwendungen sind von den Änderungen der dbExpress VCL nicht betroffen. Es gibt aber einige Methoden, Eigenschaften, Ereignisse, Konstanten und Aufzählungen, die entfernt oder durch äquivalente Funktionen ersetzt wurden.

Auch die dbExpress VCL-Komponenten, die in der obersten Schicht des Framework angeordnet sind, können für native und verwaltete Anwendungen verwendet werden.. In den VCL-Komponenten wurden geringe API-Änderungen an der Klasse TSQLConnection (Methodenänderungen), an der Klasse TSQLDataSet (neue Eigenschaft) und an Datenstrukturen (einige wurden entfernt oder ersetzt) vorgenommen. Unter dbExpress Framework-Kompatibilität finden Sie nähere Informationen.

Das dbExpress-Treiber-Framework:

- Ist vollständig in der Sprache Delphi geschrieben und ermöglicht das Schreiben von Treibern in Delphi.
- Verwendet einen streng typisierten Datenzugriff anstelle von Zeigern. Das Framework verwendet z.B. String-Typen anstatt Zeiger auf Strings.
- Verwendet einen einzigen Quelltext. Das bedeutet, dass eine einzige Kopie des Quelltextes entweder mit den nativen DCC32- oder den verwalteten DCCIL-Compilern compiliert werden kann.
- Hat nur abstrakte Basisklassen, die für Treiber, Verbindungen, Befehle, Reader usw. verwendet werden.
- Verwendet eine auf Exception basierte Fehlerbehandlung anstelle der Rückgabe von Fehlercodes.

VCL und RTL

Neue Komponenten: Die VCL (Visual Component Library) unterstützt die folgenden neuen Komponenten:

AJAX: RAD Studio unterstützt die AJAX-basierte RAD VCL für die Web-Entwicklung.

Microsoft Vista-Kompatibilität: RAD Studio stellt Komponenten, Klassen, Methoden und Eigenschaften bereit, die mit dem Erscheinungsbild des Vista-Betriebssystems kompatibel sind.

Neue VCL-Komponenten: Die VCL (Visual Component Library) unterstützt die folgenden neuen Klassen:

- TFileOpenDialog
- TFileSaveDialog
- TTaskDialog
- TCustomFileDialog
- TCustomFileOpenDialog
- TCustomFileSaveDialog
- TCustomTaskDialog
- TFavoriteLinkItem
- TFavoriteLinkItems
- TFavoriteLinkItemsEnumerator
- TFileTypeItem
- TFileTypeItems

- `TTaskDialogBaseButtonItem`
- `TTaskDialogButtonItem`
- `TTaskDialogButtons`
- `TTaskDialogButtonsEnumerator`
- `TTaskDialogProgressBar`
- `TTaskDialogRadioButtonItem`

Neue Routinen und Variablen des Speichermanagers: Die folgenden neuen Systemroutinen und -variablen wurden zur Unterstützung des Speichermanagers hinzugefügt:

- `AttemptToUseSharedMemoryManager`
- `GetMemoryManagerState`
- `GetMemoryMap`
- `GetMinimumBlockAlignment`
- `RegisterExpectedMemoryLeak`
- `SetMinimumBlockAlignment`
- `ShareMemoryManager`
- `UnregisterExpectedMemoryLeak`
- `NeverSleepOnMMThreadContention`
- `ReportMemoryLeakOnShutdown`

Siehe auch

Überblick zu MSBuild ([siehe Seite 1461](#))

Überblick über Build-Konfigurationen (Delphi) ([siehe Seite 1463](#))

Überblick über Build-Konfigurationen (C++) ([siehe Seite 1465](#))

Manager für Build-Konfigurationen ([siehe Seite 496](#))

Projektoptionen ([siehe Seite 517](#))

Verwenden der Befehlszeile zum Erzeugen eines Projekts ([siehe Seite 12](#))

Überblick zur Speicherverwaltung ([siehe Seite 1017](#))

Datei-Browser ([siehe Seite 806](#))

Überblick zur Modellierung ([siehe Seite 1446](#))

Targets-Dateien ([siehe Seite 1471](#))

Überblick über virtuelle Ordner

dbExpress-Framework

7.4 Ein erster Blick auf die IDE

Nach dem Start von RAD Studio befinden Sie sich in der integrierten Entwicklungsumgebung (Integrated Development Environment, IDE), in der verschiedene Tools und Menüs zur Verfügung stehen. In der IDE können Sie Benutzeroberflächen entwickeln, Objekteigenschaften festlegen, Quelltext verfassen und Ihre Anwendung anzeigen und verwalten.

Im Standard-Layout enthält der Desktop die Tools, die Sie am häufigsten benötigen. Über das Menü Ansicht können Sie bestimmte Tools nach Bedarf ein- und ausblenden. Nachdem Sie das Desktop-Layout nach Ihren Wünschen gestaltet haben, können Sie es speichern.

Die Tools, die in der IDE angeboten werden, variieren je nach RAD Studio-Edition. Folgende Tools stehen zur Verfügung:

- Willkommensseite
- Eingabehilfen
- Formulare
- Formular-Designer
- Tool-Palette
- Objektinspektor
- Objektablage
- Projektverwaltung
- Daten-Explorer
- Strukturansicht
- Versionsverwaltung
- Quelltext-Editor
- Datei-Browser
- Themes für Windows Vista

In den folgenden Abschnitten werden diese Tools beschrieben.

Themes für Windows Vista

Die IDE verwendet Themes aus Windows Vista oder Windows XP. Wenn Sie die klassische Darstellung der IDE und Ihrer Anwendung bevorzugen, können Sie Themes im Dialogfeld **Projekt**▸**Optionen** deaktivieren.

Willkommensseite

Wenn Sie RAD Studio öffnen, sehen Sie die Willkommensseite mit verschiedenen Links zu Entwickler-Ressourcen (Artikel zum Produkt, Schulung, Online-Hilfe usw.). Oben auf der Seite wird eine Liste der zuletzt bearbeiteten Projekte angezeigt. Über diese Liste können Sie bequem auf ein Projekt zugreifen. Wenn Sie die Willkommensseite geschlossen haben, können Sie diese jederzeit wieder öffnen, indem Sie **Ansicht**▸**Willkommens-Seite** wählen.

Eingabehilfen

Das Hauptmenü der IDE unterstützt MS Active Accessibility (MSAA). Sie können deshalb über das Startmenü von Windows auf die Eingabehilfen-Tools zugreifen (**Alle Programme**▸**Zubehör**▸**Eingabehilfen**).

Formulare

Ein Formular repräsentiert ein Fenster oder eine HTML-Seite in der Benutzeroberfläche. Während des Entwurfs wird das Formular im Designer angezeigt. Sie ziehen Komponenten von der Tool-Palette in das Formular, um das Aussehen der

Benutzeroberfläche zu bestimmen.

RAD Studio unterstützt mehrere Arten von Formularen, die in den folgenden Abschnitten beschrieben werden. Wählen Sie das Formular aus, das am besten für den beabsichtigten Anwendungsentwurf geeignet ist. Wählen Sie zwischen einer Webanwendung, die eine Funktionalität auf der Basis einer Geschäftslogik im Web enthalten soll oder einer Windows-Anwendung, die Daten verarbeiten und eine optimale Geschwindigkeit bei der Darstellung der Inhalte erzielen soll. Sie schalten zwischen dem Designer und dem Quelltext-Editor um, indem Sie auf die zugehörigen Register am unteren Rand der IDE klicken.

Um auf Formulare zuzugreifen, wählen Sie **Datei>Neu>Weitere**.

Windows Forms

Wenn Sie eine reine Windows-Anwendung für die Ausführung in einer verwalteten Umgebung erstellen möchten, verwenden Sie Windows Forms. Mit Hilfe der .NET-Klassen erstellen Sie Windows-Clients, die zwei wesentliche Vorteile haben: Erstens lassen sich in Windows-Clients Funktionen bereitstellen, die in Browser-Clients nicht zur Verfügung stehen. Zweitens setzen Sie das .NET Framework sinnvoll für die Infrastruktur ein. Windows Forms nutzen ein Programmiermodell, das auf der Einheitlichkeit des .NET Framework basiert (z.B. im Hinblick auf sichere und dynamische Anwendungs-Updates) und gleichzeitig die Funktionsfülle von GUI-Anwendungen unter Windows bieten. Sie verwenden die üblichen Windows-Steuerelemente, wie Schaltflächen, Listenfelder und Eingabefelder, und erstellen damit Ihre Windows-Anwendungen.

Um auf ein Windows Form zuzugreifen, wählen Sie **Datei>Neu>Weitere>Delphi für .NET-Projekte>Windows Forms-Anwendung**.

VCL-Formulare

Mit VCL-Formularen erstellen Sie native Anwendungen anhand von VCL-Komponenten oder Anwendungen anhand von VCL.NET-Komponenten, die im .NET-Framework ausgeführt werden.

VCL-Formulare eignen sich für Entwickler, die bereits mit der VCL vertraut sind. Sie vereinfachen außerdem das Portieren vorhandener Delphi-Anwendungen mit VCL-Steuerelementen in die .NET-Umgebung.

Um eine VCL-Formularanwendung zu erstellen, wählen Sie **Datei>Neu>Weitere>VCL-Formularanwendung**.

Formular-Designer

Der Formular-Designer (der Designer) wird automatisch im mittleren Bereich angezeigt, wenn Sie ein Formular öffnen. Die Art des Formulars bestimmt das Aussehen und die Funktionalität des Designers. Wenn Sie beispielsweise mit einem ASP.NET Web Form arbeiten, steht im Designer ein HTML-Tag-Editor zur Verfügung. Um den Designer zu aktivieren, klicken Sie am unteren Rand der IDE auf das Register Design.

Visuelle Komponenten

Visuelle Komponenten werden während des Entwurfs auf dem Formular angezeigt und sind zur Laufzeit für den Endbenutzer sichtbar. Es kann sich dabei um Schaltflächen, Beschriftungen, Symbolleisten, Listenfelder oder andere Steuerelemente handeln.

Formularvorschau

Ein Vorschausymbol unten rechts im Designer (für VCL-Formulare) zeigt die Bildschirmposition des Formulars zur Laufzeit an. Dies vereinfacht die korrekte Platzierung der Formulare während des Anwendungsentwurfs.

HTML-Designer

Mit dem HTML-Designer zeigen Sie ASP.NET Web Forms oder HTML-Seiten an bzw. bearbeiten diese. Dieser Designer enthält einen Tag-Editor für die Bearbeitung von HTML-Tags neben der visuellen Darstellung des Formulars bzw. der Seite. Sie können auch den Objektinspektor verwenden, um die Eigenschaften eines im HTML-Dokument dargestellten Elements zu bearbeiten. Im Tag-Editor werden die Eigenschaften des jeweils aktuellen HTML-Tags angezeigt. In einem Kombinationsfeld über dem Tag-Editor können SCRIPT-Tags angezeigt und bearbeitet werden.

Um eine neue HTML-Datei zu erstellen, wählen Sie **Datei**▶**Neu**▶**Weitere**▶**Web-Dokumente**▶**HTML-Seite**.

Nicht-visuelle Komponenten und die Komponentenablage

Nicht-visuelle Komponenten werden während des Entwurfs zum Formular hinzugefügt, sind aber zur Laufzeit für den Endbenutzer nicht sichtbar. Die nicht-visuellen Komponenten lassen sich für die Wiederverwendung von Datenbankgruppen und Systemobjekten einsetzen, um jene Teile einer Anwendung zu isolieren, die die Datenbankverbindung und die Geschäftsregeln behandeln.

Die nicht-visuellen Komponenten eines Formulars werden in der Komponentenablage am unteren Rand der Design-Oberfläche angezeigt. Mit Hilfe der Komponentenablage lassen sich visuelle und nicht-visuelle Komponenten einfach voneinander unterscheiden.

Designer-Richtlinien

Wenn Sie Komponenten für ein Formular erstellen, können Sie einen Objekttyp registrieren und dann verschiedene Punkte auf oder in der Nähe der Komponente bestimmen, die als Ausrichtungspunkte dienen sollen. Die Ausrichtungspunkte werden durch vertikale oder horizontale Linien definiert, die die Grenzen eines visuellen Steuerelements schneiden.

Nachdem die Ausrichtungspunkte festgelegt wurden, können Sie oberflächenspezifische Vorgaben für das Steuerelement festlegen, etwa den Abstand zum nächsten Steuerelement, Verknüpfungen, Fokusbeschriftungen, Tabulatorreihenfolge, maximale Anzahl der enthaltenen Elemente (Listen, Menüs) usw. Auf diese Weise lassen sich bewährte Richtlinien für Benutzeroberflächen bereits im Formular-Designer implementieren.

Wenn sowohl die Option Am Raster ausrichten als auch die Verwendung von Designer-Richtlinien aktiviert ist, haben die Designer-Richtlinien Vorrang. Liegt ein Rasterpunkt beispielsweise innerhalb der Toleranzgrenze und ist für denselben Abstand auch eine Richtlinie aktiv, wird das Steuerelement gemäß der Richtlinie platziert. Dies gilt auch dann, wenn die Richtlinie nicht mit dem Rasterpunkt identisch ist. Die Ausrichtungstoleranz ist von der Rastergröße abhängig. Der Designer bestimmt die Toleranz auch dann anhand der Rastergröße, wenn die Optionen Am Raster ausrichten und Raster anzeigen deaktiviert sind.

Diese Funktion steht gegenwärtig nur für VCL und VCL.NET (einschließlich C++) zur Verfügung. Für WinForms wird sie noch nicht unterstützt. Weitere Informationen zum Definieren von Designer-Richtlinien finden Sie über den entsprechenden Link am Ende dieses Themas.

Tool-Palette

Die Tool-Palette, die sich auf der rechten Seite befindet, enthält Elemente, die Sie bei der Entwicklung Ihrer Anwendung unterstützen. Die angezeigten Elemente variieren je nach Ansicht. Wenn Sie beispielsweise ein Formular im Designer anzeigen, enthält die Tool-Palette Komponenten, die für dieses Formular geeignet sind. Durch Doppelklicken auf eine Komponente können Sie das entsprechende Steuerelement in das Formular einfügen. Sie können es auch per Drag&Drop auf dem Formular platzieren. Wenn Sie Quelltext im Quelltext-Editor anzeigen, enthält die Tool-Palette Quelltextsegmente, die Sie Ihrer Anwendung hinzufügen können.

Benutzerdefinierte Komponenten

Neben den Komponenten, die bereits in RAD Studio installiert sind, können Sie benutzerdefinierte Komponenten oder Fremdkomponenten in die Tool-Palette einfügen und in eigenen Kategorien speichern.

Komponenten-Templates

Sie können Templates (Vorlagen) erstellen, die aus einer oder mehreren Komponenten bestehen. Nachdem Sie Komponenten in ein Formular eingefügt, ihre Eigenschaften festgelegt und Code für sie verfasst haben, können Sie diese Informationen in Form einer Komponenten-Template speichern. Später müssen Sie dann lediglich die Template in der Tool-Palette auswählen, um die vorkonfigurierten Komponenten in einem Schritt in das Formular zu übernehmen. Dabei werden auch alle zugehörigen Eigenschaften und Ereignisbehandlungsroutinen zum Projekt hinzugefügt. Nach dem Einfügen können Sie nach Bedarf die Position der Komponenten ändern, ihre Eigenschaften neu zuweisen und Ereignisbehandlungsroutinen erstellen oder bearbeiten.

Objektinspektor

Mit dem Objektinspektor, der sich auf der linken Seite befindet, weisen Sie den Komponenten Eigenschaften zur Entwurfszeit zu und erstellen Ereignisbehandlungs routinen für sie. Auf diese Weise entsteht die Verbindung zwischen dem visuellen Erscheinungsbild Ihrer Anwendung und dem Code, der die Anwendung zur Ausführung bringt. Der Objektinspektor enthält zwei Registerkarten: Eigenschaften und Ereignisse.

Aktivieren Sie die Registerkarte Eigenschaften, wenn Sie die physischen Attribute einer Komponente ändern möchten. Je nach Auswahl können Sie in manchen Kategorienoptionen Werte in ein Textfeld eingeben oder Werte aus einer Dropdown-Liste auswählen. Bei booleschen Operationen wechseln Sie zwischen **True** und **False**. Nachdem Sie das physische Erscheinungsbild einer Komponente definiert haben, erstellen Sie die Ereignisbehandlungs routinen, die festlegen, wie die Komponente funktioniert.

Auf der Registerkarte Ereignisse definieren Sie für das ausgewählte Objekt Ereignisse. Gibt es für dieses Objekt bereits vordefinierte Ereignisbehandlungs routinen, wählen Sie diese aus dem Dropdown-Feld aus. Im Objektinspektor sind einige Optionen standardmäßig ausgeblendet. Um diese Optionen einzublenden, klicken Sie auf das Pluszeichen neben der Kategorie.

Manche nicht-visuellen Komponenten, zum Beispiel die Borland-Datenprovider, ermöglichen einen schnellen Zugang zu bestimmten Editoren, zum Beispiel dem Verbindungseditor und dem Anweisungstext-Editor. Diese Editoren lassen sich am unteren Rand des Objektinspektors im Bereich Designer Verb aktivieren. Um einen Editor zu öffnen, setzen Sie den Mauszeiger auf den Namen des Editors, bis der Mauszeiger die Form einer Hand annimmt und der Editortname sich in einen Link verwandelt. Alternativ können Sie mit der rechten Maustaste auf eine nicht-visuelle Komponente klicken und den zugehörigen Editor aus dem Kontextmenü auswählen. Beachten Sie, dass nicht alle nicht-visuellen Komponenten über einen Editor verfügen. Neben den Editoren werden in diesem Bereich auch Hyperlinks dargestellt, die benutzerdefinierte Komponenteneditoren anzeigen, eine Webseite aufrufen oder Dialogfelder einblenden.

Objektablage

Durch vordefinierte Templates, Formulare und andere, schnell zugängliche Elemente, die sich für die Anwendung verwenden lassen, wird die Programmierung in RAD Studio sehr einfach.

Inhalt der Objektablage

Die Objektablage enthält Elemente für die verschiedenen Arten von Anwendungen, die Sie entwickeln können. Es sind auch Templates, Formulare und andere Elemente darin zu finden. Mit Hilfe der verfügbaren Templates können Sie Projekte wie Klassenbibliotheken, Steuerelementbibliotheken, Konsolenanwendungen, HTML-Seiten und anderes erstellen.

Zum Öffnen der Objektablage wählen Sie **Datei>Neu>Weitere**. Im Dialogfeld Objektgalerie wird dann der Inhalt der Objektablage angezeigt. Vorhandene Objekte lassen sich in der Objektablage bearbeiten oder daraus entfernen. Dazu klicken Sie mit der rechten Maustaste in die Objektablage, um die verfügbaren Bearbeitungsoptionen anzuzeigen.

Objektablage-Templates

Sie können in die Objektablage eigene Projekte als Templates einfügen, um diese wieder zu verwenden oder für andere Entwickler bereitzustellen. Die mehrfache Verwendung von Objekten hat den Vorteil, dass Sie Programm familien mit gemeinsamer Benutzeroberfläche und Funktionalität erstellen können. Dadurch sparen Sie nicht nur Entwicklungszeit, sondern verbessern auch die Qualität Ihrer Produkte.

Sie können der Objektablage Dateien (z.B. für ein Startprojekt, eine Demo oder eine Template) hinzufügen, die sich dann über das Menü Neu abrufen lassen. Wählen Sie **Projekt>Der Objektablage hinzufügen**. Wählen Sie die gewünschte Datei aus. Wenn Sie anschließend **Datei>Neu** wählen, können Sie eine Kopie der hinzugefügten Datei laden und mit ihr arbeiten.

Projektverwaltung

Ein Projekt besteht aus mehreren Anwendungsdateien. In der Projektverwaltung, die sich auf der rechten Seite befindet, können Sie die einzelnen Projektdaten, z.B. Formulare, ausführbare Dateien, Assemblierungen, Objekte und Bibliotheksdateien, anzeigen und organisieren. Dateien können in der Projektverwaltung hinzugefügt, gelöscht und umbenannt werden. Außerdem lassen sich damit verwandte Projekte zu einer Projektgruppe zusammenfassen, um sie zur gleichen Zeit zu compilieren.

Referenzen hinzufügen

Sie können bereits vorhandene COM-Server und ActiveX-Elemente in verwaltete Anwendungen integrieren, indem Sie Referenzen zu unverwalteten DLLs in Ihr Projekt einfügen und dann wie bei verwalteten Assemblierungen die Typen einfach durchsuchen. Wählen Sie **Projekt>Referenz hinzufügen**, um vorhandene COM-Server oder ActiveX-Elemente zu integrieren. Wahlweise klicken Sie mit rechten Maustaste auf den Ordner Referenz in der Projektverwaltung und wählen Referenz hinzufügen. Mit Referenz hinzufügen können auch andere .NET-Assemblierungen, COM/ActiveX-Komponenten oder Typbibliotheken hinzugefügt werden.

Referenzen in lokalen Pfad kopieren

Zur Laufzeit müssen sich Assemblierungen im Ausgabepfad des Projekts befinden oder zum Deployment im globalen Assemblierungs-Cache (GAC). Um eine Referenz in den lokalen Ausgabepfad zu kopieren, klicken Sie in der Projektverwaltung mit der rechten Maustaste auf eine Assemblierung und wählen die Einstellung Lokal kopieren. Beachten Sie folgende Richtlinien, um zu bestimmen, ob eine Referenz kopiert werden muss:

- Wenn sich die Referenz auf eine Assemblierung bezieht, die bereits in einem anderen Projekt erstellt wurde, aktivieren Sie die Einstellung Lokal kopieren.
- Wenn sich die Assemblierung im GAC befindet, darf Lokal kopieren nicht aktiviert werden.

Webreferenzen einfügen

In eine Client-Anwendung lassen sich Webreferenzen schnell einfügen, um auf den gewünschten Web-Service zuzugreifen. Wenn Sie eine Webreferenz hinzufügen, importieren Sie ein WSDL-Dokument, das einen bestimmten Web-Service beschreibt, in die Client-Anwendung. Sobald das WSDL-Dokument importiert ist, generiert RAD Studio alle Interfaces und Klassendefinitionen, die für den Aufruf des Web-Service erforderlich sind. Um die Funktion zum Einfügen von Webreferenzen zu nutzen, klicken Sie in der Projektverwaltung mit der rechten Maustaste auf den Knoten Web-Services.

Daten-Explorer

Mit dem Daten-Explorer können Sie serverspezifische Schemaobjekte von Datenbanken, wie Tabellen, Felder, Stored Procedure-Definitionen, Trigger und Indizes durchsuchen. Die Erstellung und Verwaltung von Datenbankverbindungen erfolgt über Kontextmenüs. Außerdem können Sie Daten aus einer Datenquelle in die meisten Formulare ziehen und so schnell Ihre Datenbankanwendung erstellen.

Strukturansicht

Die Strukturansicht enthält die hierarchische Struktur des Quelltextes bzw. der HTML-Elemente im Quelltext-Editor oder der Komponenten im Designer. Wenn Sie in der Strukturansicht für Quelltext oder HTML auf ein Element doppelklicken, wird im Quelltext-Editor die entsprechende Stelle bzw. Deklaration angezeigt. Wenn Komponenten angezeigt werden, können Sie auf eine Komponente doppelklicken, um sie im Formular auszuwählen.

Falls der Quelltext Syntaxfehler enthält, werden diese im Ordner Fehler der Strukturansicht angezeigt. Wenn Sie auf einen Fehler doppelklicken, wird die betreffende Stelle im Quelltext-Editor angezeigt.

Um den Inhalt und das Erscheinungsbild der Strukturansicht anzupassen, wählen Sie **Tools>Optionen>Umgebungsoptionen>Explorer** und ändern die Einstellungen nach Bedarf.

Versionsverwaltung

In der Ansicht Versionsverwaltung, die sich im mittleren Bereich befindet, können Sie frühere Versionen einer Datei anzeigen und miteinander vergleichen. In dieser Ansicht lassen sich z.B. verschiedene Sicherungsversionen, gespeicherte lokale Änderungen oder der Puffer mit nicht gespeicherten Änderungen für die aktive Datei anzeigen. Unterliegt die aktuelle Datei der Versionskontrolle, sind alle Revisionstypen in der Versionsverwaltung zu sehen. Die Ansicht Versionsverwaltung befindet sich rechts neben der Registerkarte Code und enthält die folgenden Registerkarten (Seiten):

- Auf der Seite Inhalt werden die aktuelle Version und frühere Versionen der Datei angezeigt.
- Auf der Seite Unterschied werden die Unterschiede zwischen den ausgewählten Versionen der Datei dargestellt.
- Die Seite Info enthält alle Beschriftungen und Kommentare der aktiven Datei.

Mit den Schaltflächen in der Symbolleiste der Versionsverwaltung können Sie Revisionsinformationen aktualisieren, eine Version auf den Stand der neusten Version bringen und den Bildlauf zwischen der Quelltextanzeige der Seite Inhalt oder Unterschied mit derjenigen des Quelltext-Editors synchronisieren sowie Navigationsoperationen (wie zum nächsten Unterschied springen) ausführen.

Quelltext-Editor

Im Quelltext-Editor, der sich im mittleren Bereich befindet, können Sie den Quelltext Ihrer Anwendungen anzeigen und bearbeiten. Dieser leistungsstarke UTF8-Editor unterstützt das Refactoring von Quelltext, automatische Sicherungen, eine Programmierhilfe, die Syntaxhervorhebung, das Rückgängigmachen mehrerer Aktionen, eine kontextsensitive Hilfe, Quelltext-Templates, die automatische Blockvervollständigung, die Suche nach Klassen und Units, das Importieren von Namespaces und vieles mehr. Über den entsprechenden Link am Ende dieses Themas finden Sie weitere Informationen zu den Funktionen und Leistungsmerkmalen des Quelltext-Editors.

Datei-Browser

Mit dem Datei-Browser, einem andockbaren Browser-Fenster im Windows-Stil, können Sie grundlegende Dateioperationen ausführen. Öffnen Sie den Datei-Browser, um eine Datei zu suchen, umzubenennen oder Versionskontrolloperationen für eine Datei auszuführen.

Zum Ausführen einer Operation mit einer Datei wählen Sie **Ansicht**▶**Datei-Browser**, navigieren zu der Datei, klicken sie mit der rechten Maustaste an und wählen die gewünschte Operation aus.

Siehe auch

Ein Projekt starten (siehe Seite 1369)

Formular-Designer

Komponenten in ein Formular einfügen (siehe Seite 55)

Komponenteneigenschaften festlegen (siehe Seite 73)

Quelltext-Editor (siehe Seite 1373)

Quelltext-Editor anpassen (siehe Seite 44)

Windows Forms-Anwendungen erstellen

Deaktivieren von Windows Vista- oder Windows XP-Themes (siehe Seite 65)

Datei-Browser (siehe Seite 806)

Mit der Versionsverwaltung arbeiten (siehe Seite 52)

7.5 Ein Projekt starten

Ein Projekt ist eine Sammlung von Dateien, aus denen eine Zielanwendung erstellt wird. Die Sammlung umfasst neben den von Ihnen hinzugefügten und bearbeiteten Dateien (z.B. Quelltext- und Ressourcendateien) auch Dateien, die RAD Studio zum Speichern von Projekteinstellungen verwendet (z.B. die `.dproj`-Projektdatei). Projekte werden zur Entwurfszeit erstellt und liefern bei der Compilierung die Projekt-Zieldateien (.exe, .dll, .bpl usw.). Um Sie beim Entwicklungsprozess zu unterstützen, sind in der Objektablage viele vordefinierte Templates, Formulare, Dateien und andere Elemente enthalten, auf die Sie zurückgreifen können.

Sie erstellen ein Projekt, indem Sie auf der Willkommensseite auf Neu klicken und den Anwendungstyp auswählen, den Sie erstellen möchten. Sie können alternativ auch **Datei>Neu>Weitere** wählen. Um ein bereits vorhandenes Projekt zu öffnen, klicken Sie auf der Willkommensseite auf Projekt oder wählen alternativ **Datei>Projekt öffnen**.

Dieser Abschnitt enthält Informationen über:

- Projektarten
- Arbeit mit nicht verwaltetem Code

Projektarten

Je nachdem, mit welcher Edition von RAD Studio Sie arbeiten, können Sie reine Windows-Anwendungen, ASP.NET-Webanwendungen, ADO.NET-Datenbankanwendungen, Web-Services-Anwendungen und viele andere erstellen. RAD Studio unterstützt auch Assemblierungen, benutzerdefinierte Komponenten, Multithreading und COM. Weitere Informationen zu den verfügbaren Tools in den einzelnen Editionen finden Sie in der Funktionsmatrix auf der CodeGear Delphi-Webseite oder der CodeGear C#Builder-Webseite.

Windows-Anwendungen

Reine Windows-Anwendungen erstellen Sie mit Windows Forms, um eine leistungsstarke Verarbeitung und die Anzeigefunktion für die Inhalte zu nutzen. Neben den traditionellen Einsatzmöglichkeiten können Windows-Anwendungen auch mit Konstrukten des neueren .NET Framework verwendet werden. Beispielsweise ist es möglich, eine Windows-Anwendung als Frontend für eine ADO.NET-Datenbank zu benutzen.

ASP.NET-Webanwendungen

Webanwendungen erstellen Sie mit ASP.NET Web Forms und binden damit einen Web-Zugang zu Datenbanken und Web-Services ein. Web Forms enthalten Benutzeroberflächen für Webanwendungen und bestehen aus HTML und Server Controls. Die Anwendungslogik wird in den Codebehind-Dateien bereitgestellt. In RAD Studio können Sie die Komponenten per Drag&Drop ziehen und den HTML-Code an der gewünschten Position bearbeiten.

Neben den Drag&Drop-Komponenten und den visuellen Designern bietet CodeGear eine schnelle Möglichkeit, die Menüs und Untermenüs einer Anwendung zu erstellen. Die Komponenten MainMenu und ContextMenu des .NET Menü-Designers lassen sich wie Editoren bedienen. Sie können damit auf visueller Ebene Menüs entwerfen und den Code für deren Funktionalität schnell hinzufügen.

ASP.NET-Web-Services-Anwendungen

Sie können Web-Services-Anwendungen erstellen, die Inhalte wie HTML-Seiten oder XML-Dokumente über das Web bereitstellen. Web-Services ist eine Internet-basierte Integrationstechnik, die es ermöglicht, verschiedene Anwendungen via Web zu verbinden und Informationen über Standard-Nachrichtenprotokolle auszutauschen.

RAD Studio vereinfacht die Erstellung von Web-Services, indem es einfache Methoden zum Erstellen von SOAP-Server-Anwendungen bereitstellt. Die `.asmx`- und `.dll`-Dateien werden dabei automatisch erstellt. Ein Web-Service

lässt sich innerhalb der IDE testen, ohne eine Client-Anwendung dafür erstellen zu müssen.

Wenn Sie eine Client-Anwendung schreiben, die einen öffentlichen Web-Service verwendet, können Sie den UDDI-Browser dazu benutzen, die WSDL zu suchen und zu importieren, die den Web-Service in Ihrer Client-Anwendung beschreibt.

VCL .NET-Anwendungen

Verwenden Sie VCL-Formulare, um eine .NET Windows-Anwendung zu erstellen, die Komponenten aus dem VCL.NET Framework verwendet.

RAD Studio vereinfacht das Erstellen von .NET-fähigen Anwendungen, denn es unterstützt VCL-Komponenten, die für die Ausführung im .NET Framework erweitert wurden. Damit müssen Sie keine benutzerdefinierten Komponenten mehr entwickeln, um die Funktionalität der VCL-Standardkomponenten zu nutzen. Die Portierung von Win32-Anwendungen in die .NET-Umgebung ist damit einfach und stabil geworden.

Datenbankanwendungen

Gleichgültig, ob Ihre Anwendungen Windows Forms, Web Forms oder VCL-Formulare verwenden, RAD Studio verfügt über die notwendigen Tools, um eine Datenbankverbindung herzustellen, eine Datenbank zu durchsuchen, zu bearbeiten, SQL-Abfragen durchzuführen und die aktuellen Daten live zur Entwurfszeit anzuzeigen.

Mit dbExpress können Sie eine Verbindung zu Interbase-, Oracle-, MS SQL-, Informix-, DB2-, Sybase- und MySQL-Datenbanken herstellen. Sie können auch Datenbanktreiber entwickeln, indem Sie die Klassen des dbExpress-Frameworks erweitern. Mit dbExpress lässt sich sowohl nativer als auch verwalteter Code verwenden

Die Datenprovider des ADO.NET Framework bieten Zugang zu MS SQL-, Oracle- und ODBC- sowie OLE DB-kompatiblen Datenbanken. Mit den Borland-Datenprovidern (BDP.NET) erhalten Sie Zugang zu MS SQL-, Oracle-, DB2- und InterBase-Datenbanken. Sie können zu jeder dieser Datenquellen eine Verbindung herstellen, deren Daten in Datenmengen übernehmen und die erforderlichen SQL-Anweisungen zur Datenbearbeitung benutzen. Die Verwendung von BDP.NET bietet folgende Vorteile:

- Portierbarer Code, der einmal geschrieben werden muss und die Verbindung zu beliebigen unterstützten Datenbanken herstellt.
- Eine offene Architektur, die Unterstützung für zusätzliche Datenbanksysteme ermöglicht.
- Eine Zuordnung logischer Datentypen zu nativen .NET-Typen.
- Eine konsistente Datentypzuordnung zwischen den verschiedenen Datenbanken (sofern verfügbar).
- Anders als bei OLE DB ist keine COM Interop-Schicht erforderlich.

Wenn Sie VCL-Formulare und die VCL.NET-Framework-Komponenten verwenden, können Sie die Unterstützung für Datenbanken noch erweitern, indem Sie BDE.NET, dbExpress.NET und den Midas Client für .NET-Verbindungstechniken einsetzen.

Modellgesteuerte Anwendungen

Der gesamte Prozess des Software-Entwurfs wird als Modellierung bezeichnet. Ein Modell für ein Softwaresystem ist gut mit den Modellen zu vergleichen, die ein Architekt für ein Gebäude anfertigt. Ein Modell stellt ein System nicht nur im Ganzen dar, sondern bietet auch die Möglichkeit, sich auf spezielle Aspekte wie die Struktur und Details im Verhalten zu konzentrieren. Unabhängig von der jeweiligen Programmiersprache (und auch von bestimmten Technologien) können alle am Entwicklungszyklus Beteiligten mit Hilfe des Modells jederzeit in derselben Sprache miteinander kommunizieren.

Die modellgesteuerte Architektur (Model Driven Architecture, MDA) von CodeGear stellt einen Ansatz für das Software-Engineering dar, bei dem die Modellierungstools in die Entwicklungsumgebung selbst integriert sind. Die MDA wurde im Umfeld des CodeGear Enterprise Core Objects (ECO) Framework entwickelt. Das ECO-Framework besteht aus einer Sammlung von Interfaces, Klassen und benutzerdefinierten Attributen, die für die Kommunikation zwischen Ihrer Anwendung und den Modellierungsfunktionen der IDE sorgen.

Zu den ECO-Funktionen gehören:

- Automatische Konvertierung der Modellklassen mit ihren Attributen und Beziehungen in ein relationales Schema.
- Automatische Weiterentwicklung des Schemas bei einer Änderung des Modells.
- Spezifikation des Persistenz-Backends. Sie können Objekte entweder in einer relationalen Datenbank oder in einer XML-Datei speichern.
- Zur Entwurfszeit wird eine strukturelle Gültigkeitsprüfung des Modells und seiner Object-Constraint-Language-(OCL)-Ausdrücke durchgeführt.
- Die OCL-Ausdrücke werden zur Laufzeit auf Gültigkeit überprüft.
- Es steht ein Ereignismechanismus zur Verfügung, der entsprechende Mitteilungen an Sie versendet, wenn Objekte hinzugefügt, verändert oder entfernt werden.

Die IDE von RAD Studio nutzt das ECO-Framework, um für die Entwicklung des Anwendungsmodells eine integrierte Oberfläche zu bieten. Zu den Modellierungsfunktionen der IDE gehören:

- Erstellen modellgesteuerter Anwendungen als eine neue Art Projekt.
- Erstellen und Bearbeiten von Klassendiagrammen und Modellelementen (Packages und Klassen) direkt auf der Oberfläche.
- Einfügen, Entfernen und Ändern von Klassenattributen und Methoden in Klassendiagrammen.
- Wechselseitige Aktualisierung zwischen Quelltext und Modellierungsoberfläche. Änderungen im Quelltext werden in der grafischen Abbildung dargestellt und umgekehrt.
- Beidseitige Navigationsmöglichkeit zwischen Modellelementen und Quelltext. Sie können von der grafischen Abbildung eines Modellelements direkt zum zugehörigen Quelltext navigieren. Ähnlich können Sie auch von einer modellierten Klasse im Quelltext direkt zum entsprechenden Diagramm auf der Modellierungsoberfläche navigieren.
- Export und Import von Modellen mit XMI 1.1.

Anmerkung: Einige Modellierungsfunktionen stehen in bestimmten Editionen von RAD Studio nicht zur Verfügung. Informationen zu den Modellierungsfunktionen, die in Ihrer Edition zur Verfügung stehen, finden Sie in der Funktionsmatrix auf der CodeGear Delphi-Webseite oder der CodeGear C#Builder-Webseite.

Assemblierungen

Eine Assemblierung ist ein logisches Paket, ähnlich einer DLL-Datei, die aus Manifesten, Modulen, portierbaren Anwendungsdateien (PE) und Ressourcen ([.html](#), [.jpeg](#), [.gif](#)) besteht und zum Deployment und zur Versionierung verwendet wird. Eine Anwendung kann über eine oder mehrere Assemblierungen verfügen, die wiederum von einer oder mehreren Anwendungen referenziert werden kann, je nachdem, ob die Assemblierungen sich in einem Anwendungsverzeichnis befinden oder in dem globalen Assemblierungs-Cache (GAC).

Zusätzliche Projekte

Zusätzlich zu den oben beschriebenen Projekttypen enthält RAD Studio Templates für Klassenbibliotheken, Steuerelementbibliotheken, Konsolenanwendungen, Visual Basic-Anwendungen, Berichte, Textdateien und anderes. Diese Templates befinden sich in der Objektablage und können mit **Datei>Neu>Weitere** geöffnet werden.

Nicht verwalteter Code und COM/Interop

Nicht verwalteter Code eignet sich für Anwendungen, die nicht auf die allgemeine Laufzeitumgebung (CLR) des .NET Framework abzielen. COM Interop ist ein .NET-Service, der eine problemlose Zusammenarbeit mit verwaltetem und nicht verwaltetem Code ermöglicht. Der Service COM/Interop ermöglicht die Einbindung bereits vorhandener COM-Server und ActiveX-Elemente in Ihre .NET-Anwendungen und stellt .NET-Komponenten in vorhandenen, nicht verwalteten Anwendungen bereit. Die IDE von RAD Studio enthält Tools, die Sie bei der Integration vorhandener COM-Server und ActiveX-Elemente in verwaltete Anwendungen unterstützen. Ferner lassen sich Referenzen zu nicht verwalteten DLLs in ein Projekt einfügen. Anschließend können Sie ebenso wie bei verwalteten Assemblierungen die enthaltenen Typen durchsuchen.

Siehe auch

[Ein Projekt erstellen \(siehe Seite 61\)](#)

[Windows Forms-Anwendung erstellen](#)

[ASP.NET-Anwendungen erstellen](#)

[ASP.NET-Web-Services-Anwendung "Hello World" erstellen](#)

[Datenbankanwendung mit Windows Forms erstellen](#)

[Referenzen zu einem COM-Server hinzufügen](#)

7.6 Quelltext-Editor

Der Quelltext-Editor befindet sich im mittleren Bereich des IDE-Fensters. Der Quelltext-Editor ist ein leistungsstarker und konfigurierbarer UTF8-Editor. Er unterstützt eine Syntaxhervorhebung, das Rückgängigmachen mehrerer Aktionen und eine kontextsensitive Hilfe für Sprachelemente.

Während Sie die Benutzeroberfläche einer Anwendung entwerfen, erzeugt RAD Studio den zugrunde liegenden Quelltext. Änderungen, die Sie an den Objekteigenschaften vornehmen, werden automatisch in die Quelldateien übernommen.

Da alle Ihre Programme bestimmte gemeinsame Merkmale aufweisen, erstellt RAD Studio bestimmten Quelltext automatisch, um Ihnen den Start zu erleichtern. Sie können sich den automatisch erzeugten Quelltext als eine Art Gliederung vorstellen, die Ihnen bei der Erstellung eines Programms Orientierung bietet.

Hinweis: Wenn Sie WinForms verwenden, dürfen Sie den automatisch erzeugten Quelltext der Methode *InitializeComponents* nicht ändern, da andernfalls das Formular nicht mehr angezeigt wird, wenn Sie auf das Register Design klicken.

Der Quelltext-Editor verfügt über folgende Leistungsmerkmale, die Sie bei der Programmentwicklung unterstützen:

- Änderungsbalken
- Code Insight
- Programmierhilfe
- Quelltext-Browser
- Symbolbeschreibung
- Quelltext-Templates
- Code-Folding
- Refactoring
- Sync-Bearbeitungsmodus
- To-Do-Listen
- Tastatur-Makros
- Positionsmarken
- Kommentarblöcke

Änderungsbalken

Auf der linken Seite des Quelltext-Editors wird vor den Zeilen ein grüner Änderungsbalken angezeigt, die in der aktuellen Bearbeitungssitzung nicht geändert wurden. Ein gelber Änderungsbalken zeigt an, dass seit dem letzten Speichern Änderungen vorgenommen wurden.

Sie können aber die Standardfarben der Änderungsbalkens (grün und gelb) nach Bedarf anpassen.

Code Insight

Als Code Insight wird eine bestimmte Untergruppe von Funktionen bezeichnet, die in den Quelltext-Editor eingebettet ist und dazu dient, die Eingabe von Quelltext zu erleichtern (Code-Parameter-Hinweise, Code-Hinweise, Symbolbeschreibung, Programmierhilfe, Klassenvervollständigung, Blockvervollständigung und Quelltext-Browser). Diese Funktionen unterstützen Sie bei der Erkennung häufig verwendeter Anweisungen, die in den Quelltext eingefügt werden sollen, und bei der Auswahl von Eigenschaften und Methoden. Auf einige dieser Funktionen wird in den folgenden Abschnitten detaillierter eingegangen.

Sie können Code Insight aufrufen, indem Sie im Quelltext-Editor die Tastenkombination STRG+LEER drücken. In einem Popup-Fenster wird dann eine Liste der Symbole angezeigt, die an der Cursorposition zulässig sind.

Um die Funktionen von Code Insight zu aktivieren oder einzustellen, wählen Sie **Tools**▶**Optionen**▶**Editor-Optionen** und klicken auf Code Insight.

Bei der Programmierung in Delphi werden im Popup-Fenster keine Deklarationen von Interface-Methoden angezeigt, auf die in Lese- und Schreibroutinen für Eigenschaften Bezug genommen wird. Die Liste enthält nur Eigenschaften und eigenständige Methoden, die im Interface-Typ deklariert sind. Code Insight unterstützt WM_xxx-, CM_xxx- und CN_xxx-Botschaftsmethoden, basierend auf gleichnamigen Konstanten, die in den in der **uses**-Klausel angegebenen Units enthalten sind.

Code-Parameter-Hinweise

Diese Funktion zeigt einen Hinweis mit den Argumentnamen und -typen von Methodenaufrufen an. Sie steht zur Verfügung, wenn sich der Cursor zwischen den Klammern eines Aufrufs befindet, z.B. `ShowMessage ()`;

Um die Code-Parameter-Hinweise zu aktivieren, drücken Sie **STRG+UMSCHALT+LEER**.

Quelltext-Hinweise

Sie können einen Hinweis mit Informationen zum aktuellen Symbol anzeigen (Typ, Datei, Zeilennummer der Deklaration).

Der Quelltext-Hinweis wird angezeigt, wenn Sie den Mauszeiger im Quelltext-Editor über einem Bezeichner ruhen lassen.

Anmerkung: Quelltext-Hinweise werden nur angezeigt, wenn die Symbolbeschreibung deaktiviert ist.

Symbolbeschreibung

Sie können Informationen zum aktuellen Symbol anzeigen, z.B. den Typ, die Datei, die Zeilennummer mit der Deklaration und die verknüpfte XML-Dokumentation (falls vorhanden).

Die Symbolbeschreibung wird eingeblendet, wenn Sie den Mauszeiger im Quelltext-Editor über einem Bezeichner ruhen lassen. Sie können die Symbolbeschreibung auch durch Drücken von **STRG+UMSCHALT+H** aktivieren.

Programmierhilfe

Mit der Funktion zur Programmierhilfe können Sie eine Liste der Symbole anzeigen, die an der aktuellen Cursorposition verfügbar sind. Sie können die Programmierhilfe folgendermaßen aktivieren:

Delphi: **STRG + LEER + .**

C#: **STRG + LEER + .**

C++: **STRG + LEER + —>**

Klassenvervollständigung

Die Klassenvervollständigung vereinfacht das Definieren und Implementieren neuer Klassen, indem für die zu deklarierenden Klassenelemente Skeleton-Quelltext generiert wird. Wenn Sie den Cursor in eine Klassendeklaration im **interface**-Abschnitt einer Unit setzen und **STRG+UMSCHALT+C** drücken, werden alle noch nicht abgeschlossenen Eigenschaftsdeklarationen automatisch vervollständigt. Für alle Methoden, die eine Implementierung erfordern, werden leere Methoden in den **implementation**-Abschnitt eingefügt. Eine Auswahl ist auch über das Kontextmenü des Editors möglich.

Blockvervollständigung

Wenn Sie im Quelltext-Editor **EINGABE** drücken, und ein Quelltextblock vorhanden ist, der nicht korrekt geschlossen wurde, wird in der nächsten leeren Zeile nach der aktuellen Cursorposition der Bezeichner für das Schließen des Blocks eingefügt. Wenn Sie beispielsweise für ein Delphi-Programm im Quelltext-Editor **begin** eingeben und anschließend **EINGABE** drücken, wird die Anweisung automatisch vervollständigt. Sie lautet dann **begin end;**. Diese Funktion ist auch für die Sprachen C# und C++ verfügbar.

Quelltext-Browser

Wenn Sie eine VCL-Formularanwendung bearbeiten, können Sie im Quelltext-Editor die STRG-Taste gedrückt halten, während Sie die Maus über den Namen einer Klasse, Variable, Eigenschaft, Methode oder über einen anderen Bezeichner bewegen. Der Mauszeiger nimmt die Form einer Hand an, und der Bezeichner wird hervorgehoben und unterstrichen dargestellt. Wenn Sie auf diesen klicken, wird die Deklaration des Bezeichners im Quelltext-Editor angezeigt, wozu ggf. auch die Quelltextdatei geöffnet wird. Derselbe Vorgang lässt sich auch ausführen, indem Sie mit der rechten Maustaste auf einem Bezeichner klicken und dann Deklaration suchen wählen.

Der Quelltext-Browser kann nur Unit-Dateien finden und öffnen, die im Suchpfad oder Quelltextpfad des Projekts enthalten sind oder sich im globalen Suchpfad oder Bibliothekspfad des Produkts befinden. Verzeichnisse werden in der folgenden Reihenfolge durchsucht:

1. Suchpfad des Projekts ([Projekt](#)▶[Optionen](#)▶[Verzeichnisse/Bedingungen](#)).
2. Quelltextpfad des Projekts (das Verzeichnis, in dem das Projekt gespeichert wurde).
3. Globaler Suchpfad ([Tools](#)▶[Optionen](#)▶[Bibliothek](#)).
4. Globaler Bibliothekspfad ([Tools](#)▶[Optionen](#)▶[Umgebungsoptionen](#)▶[Delphi-Optionen](#)▶[Bibliothek](#) für Delphi).

Der Bibliothekspfad wird nur durchsucht, wenn in der IDE kein Projekt geöffnet ist.

Navigation durch Quelltext

In den folgenden Abschnitten werden die Möglichkeiten für die Navigation im Quelltext-Editor beschrieben.

Methoden-Navigation

Für die Navigation von Methode zu Methode stehen verschiedene Tastaturkürzel zur Verfügung. Es ist möglich, die Navigationsfunktion auf die Methoden der aktuellen Klasse zu beschränken. Wenn die Klassensperre (Class Lock) aktiviert ist, und sich der Cursor z.B. auf einer Methode von *TComponent* befindet, kann nur zwischen Methoden von *TComponent* navigiert werden.

Folgende Tastaturkürzel für die Methodennavigation stehen zur Verfügung:

- STRG+Q¹L – Aktivieren/Deaktivieren der Klassensperre.
- STRG+ALT+Bild oben – Sprung zum Anfang der aktuelle Methode oder zur vorherigen Methode.
- STRG+ALT+Bild unten – Sprung zur nächsten Methode.
- STRG+ALT+Pos1 – Sprung zur ersten Methode im Quelltext.
- STRG+ALT+Ende – Sprung zur letzten Methode im Quelltext.
- STRG+ALT+Mausrad – Blättern durch Methoden.

Klassen suchen

Diese Funktion ermöglicht das schnelle Auffinden von Klassen (mit regulären Ausdrücken in C#). Wenn Sie [Suchen](#)▶[Klasse suchen](#) wählen, wird eine Liste aller im Projekt verwendeten Klassen angezeigt. Wenn Sie eine Klasse auswählen, wird automatisch auf deren Deklaration positioniert.

Units suchen

Je nach der verwendeten Programmiersprache können Sie mit einer Refactoring-Funktion nach Namespaces oder Units suchen. Wenn Sie C# verwenden, können Sie mit dem Befehl Namespace importieren Namespaces in Ihren Quelltext importieren. In Delphi können Sie mit dem Befehl Unit suchen im Quelltext nach Units suchen und diese zur Quelltextdatei hinzufügen. Bei Quelltext, der auf dem .NET Framework basiert, wird der Assemblierungs-Browser geöffnet, wenn ein Ausdruck nicht vorhanden ist. Im Assemblierungs-Browser können Sie dann nach einem Typ suchen. Im Fenster für die Typsuche sind reguläre Ausdrücke

zulässig.

Quelltext-Templates

Quelltext-Templates ermöglichen es Ihnen, eine Sammlung vorgefertigter Quelltextsegmente zu erstellen, die während der Arbeit im Quelltext-Editor eingefügt werden können. Dadurch reduziert sich Ihre Schreibarbeit.

Über die Links am Ende dieses Themas finden Sie weitere Informationen zur Erstellung und Verwendung von Quelltext-Templates.

Code-Folding

Die Code-Folding-Funktion ermöglicht das Ausblenden von Codeabschnitten und die Darstellung des Quelltexts in hierarchischer Form. Dies erleichtert das Lesen und Navigieren in großen Dateien. Der ausgeblendete Code wird nicht gelöscht, sondern nur verborgen. Verwenden Sie zum Aus- und Einblenden das Minus- bzw. Plusssymbol neben den jeweiligen Codeabschnitten.

Refactoring

Beim Refactoring wird Ihr Quelltext ohne Veränderung der externen Funktionalität verbessert. Sie können beispielsweise ein ausgewähltes Quelltextfragment mit dem Befehl Methode extrahieren in eine Methode umwandeln. Die IDE verschiebt den extrahierten Code an eine Position außerhalb der aktuellen Methode, legt die erforderlichen Parameter fest, erzeugt nach Bedarf lokale Variablen, bestimmt den Rückgabetyp und ersetzt das Codefragment durch einen Aufruf der neuen Methode. Es stehen außerdem eine Reihe anderer Refactoring-Optionen, wie z.B. das Umbenennen eines Symbolen oder das Deklarieren von Variablen, zur Verfügung.

Sync-Bearbeitungsmodus

Im Sync-Bearbeitungsmodus lassen sich Bezeichner, die mehrfach im Quelltext vorhanden sind, gleichzeitig bearbeiten. Sobald Sie den ersten Bezeichner ändern, wird dieselbe Änderung automatisch an den anderen Bezeichnern durchgeführt. Es lassen sich auch Sprungpunkte setzen, um zu speziellen Abschnitten in Ihrem Quelltext zu navigieren.

To-Do-Listen

In einer To-Do-Liste werden die Programmierarbeiten verwaltet, die in einem Projekt noch erledigt werden müssen. Nachdem Sie eine Aufgabe in die To-Do-Liste eingefügt haben, können Sie sie bearbeiten, als Kommentar in den Quelltext einfügen, als erledigt markieren oder aus der Liste entfernen. Sie können die Liste auch filtern und nur bestimmte Aufgaben anzeigen.

Tastatur-Makros

Sie können eine Abfolge von Tastenanschlägen als Makro aufzeichnen, während Sie Quelltext bearbeiten. Nach der Aufzeichnung des Makros können die aufgezeichneten Tastenanschläge während der aktuellen Arbeitssitzung in der IDE beliebig oft wiederholt werden. Das neu aufgezeichnete Makro ersetzt das zuvor aufgezeichnete Makro.

Positionsmarken

Positionsmarken erleichtern die Navigation in langen Dateien. Sie können an einer Stelle im Quelltext eine Positionsmarke einfügen und später von jeder beliebigen Stelle der Datei aus zu dieser Marke springen. In einer Datei können bis zu zehn Positionsmarken (mit den Nummern 0 bis 9) gesetzt werden. Wenn Sie eine Positionsmarke setzen, wird in der linken Leiste des Quelltext-Editors ein Positionsmarkensymbol  angezeigt.

Kommentarblöcke

Sie können einen Codeblock in einen Kommentar konvertieren, indem Sie ihn im Quelltext-Editor auswählen und dann **STRG+/** (Schrägstrich) drücken. Dadurch werden jeder Codezeile des ausgewählten Blocks die Kommentarzeichen `//` vorangestellt, und der gesamte Block wird vom Compiler ignoriert. Die Tastenkombination **STRG+/** ist ein Umschalter, der Kommentarzeichen entweder hinzufügt oder entfernt. Die Wirkung hängt davon ab, ob sich vor der ersten Codezeile bereits `//` befinden. Bei Verwendung der Visual Studio- oder Visual Basic-Tastaturbelegung müssen Kommentarzeichen mit **STRG+K+C** hinzugefügt und entfernt werden.

Siehe auch

- Quelltext-Editor anpassen ( siehe Seite 44)
- Code Insight verwenden ( siehe Seite 48)
- Klassenvervollständigung verwenden ( siehe Seite 47)
- Quelltext-Templates verwenden ( siehe Seite 50)
- Quelltext-Templates erstellen ( siehe Seite 43)
- Sync-Bearbeitungsmodus verwenden ( siehe Seite 54)
- Code-Folding verwenden ( siehe Seite 42)
- To-Do-Listen verwenden ( siehe Seite 82)
- Tastatur-Makro aufzeichnen ( siehe Seite 45)
- Positionsmarken verwenden ( siehe Seite 46)

7.7 Hilfe zur Hilfe

Die mit Together ausgelieferten Hilfethemen unterstützen Sie bei der Ausführung Ihrer Aufgaben. Sie können den Inhalt der Hilfe filtern und so nur die für Ihren Produkttyp spezifischen und relevanten Informationen anzeigen.

Online-Hilfe

Die Online-Hilfe von RAD Studio besteht aus Begriffserklärungen, Anleitungen und Referenzinformationen. Die Hilfe ist so aufgebaut, dass Sie über allgemeine Erläuterungen hin zu detaillierten Informationen geführt werden.

Die Navigationsbereiche des Hilfefensters unterstützen Sie beim Auffinden der gewünschten Informationen. Standardmäßig ist kein Filter gesetzt, sodass Sie Zugriff auf den gesamten Inhalt des Hilfesystems haben. Mit der Dropdown-Liste Filter nach in den Bereichen Inhalt, Suchen und Index können Sie die Suche eingrenzen. Zur Anzeige der Navigationsbereiche verwenden Sie den Menübefehl **Ansicht**►**Navigation**.

Tip: Wenn Sie über einen Link zu einem neuen Thema gelangen, ist dessen Kontext möglicherweise nicht sofort ersichtlich. Um die Position des Themas im Inhaltsverzeichnis anzuzeigen, klicken Sie in der Symbolleiste des Hilfefensters auf die Schaltfläche Inhalt synchronisieren .

7

Begriffsübersichten

In den Begriffsübersichten werden allgemeine Informationen zur Produktarchitektur, den Komponenten und den Tools angeboten, die die Programmierung vereinfachen.

Am Ende der meisten Übersichtsbildschirme finden Sie Links zu verwandten Anleitungen und Referenzinformationen oder zu detaillierteren Informationen. Ein Symbol kennzeichnet, ob der Link zu einem Thema, zu einem Dokument eines CodeGear-Partners oder zu einer Website führt. Die Symbole werden an späterer Stelle dieses Themas erläutert.

Anleitungen

Die Anleitungen sind schrittweise aufgebaut. Für Entwicklungsaufgaben, die mehrere Teilaufgaben umfassen, sind *Basisanleitungen* verfügbar, in denen alle erforderlichen Entwicklungsschritte beschrieben werden. Wenn Sie ein neues Projekt beginnen und wissen möchten, welche Schritte ausgeführt werden müssen, lesen Sie zunächst die entsprechende Basisanleitung.

Neben den Basisanleitungen stehen verschiedene Anleitungen für Einzelaufgaben zur Verfügung.

Sämtliche Anleitungen sind unter **Anleitungen** im Bereich Inhalt des Hilfefensters zu finden.

Referenzthemen

Referenzthemen enthalten detaillierte Informationen zu wichtigen Bereichen, wie Konfigurationsoptionen, GUI-Elementen oder der Sprachreferenz.

Sämtliche Referenzthemen sind unter **Referenz** im Bereich Inhalt des Hilfefensters zu finden.

Kontextsensitive Hilfe

Die kontextsensitive Hilfe steht in der gesamten Benutzeroberfläche zur Verfügung. Sie müssen lediglich ein Element markieren und F1 oder die Schaltfläche Hilfe drücken, um den zugehörigen Hilfetext anzuzeigen.

In der Hilfe verwendete typographische Konventionen

In der gesamten Online-Hilfe von RAD Studio werden die folgenden typographischen Konventionen verwendet.

Typographische Konventionen

Konvention	Verwendet für
Schreibmaschinenschrift	Quelltext, Datei- und Ordnernamen sowie Text, der eingegeben werden soll.
Fettschrift	Reservierte Schlüsselwörter, GUI-Elemente und Dialogfelder.
<i>Kursivschrift</i>	RAD Studio-Bezeichner wie Variablen oder Typnamen. Kursiver Text wird auch für Buchtitel und zur Hervorhebung neuer Begriffe verwendet.
GROSSBUCHSTABEN	Tasten wie STRG oder EINGABETASTE.
 WEB	Ein Link zu einer Web-Ressource.
 SDK	Ein externer Link zur Microsoft SDK-Dokumentation.
 CodeGear	Ein externer Link zur Dokumentation eines CodeGear-Partners.

Siehe auch

[Together Dokumentation](#)

7.8 Erste Schritte mit Together

Zwei Beispielprojekte erleichtern Ihnen das Kennenlernen der Funktionen und Leistungsmerkmale von Together. Hierzu gehören z.B. die UML-Modellierung, die Unterstützung von Pattern und die Generierung der Projektdokumentation.

7.8.1 Allgemeines zu Together

Willkommen bei **CodeGear® Together®**, der entwurfsgesteuerten Entwicklungsumgebung für die Anwendungsmodellierung. Together bietet folgende Leistungsmerkmale: Unterstützung für UML 2.0, OCL, Pattern, Audits und Metriken für die Qualitätssicherung, Quelltextgenerierung und -Refactoring, Import von IBM Rational Rose-Modellen (MDL), XMI-Import und -Export und automatisierte Generierung von Dokumentation.

LiveSource™, eines der wichtigsten Features von Together, synchronisiert Ihre Together-Diagramme mit dem Quelltext im RAD Studio-Editor.

Together ist ein integraler Bestandteil der Komplettlösung, die die Borland Software Corporation für das Application Lifecycle Management (ALM) zur Verfügung stellt. Die vorliegende Version von Together gehört zur neuen Generation einer Borland-ALM-Lösung mit dem Namen SDO (Software Delivery Optimization). SDO bietet einen integrierten und stringenten Ansatz für die Softwarebereitstellung, bei der Teams, Technologien und Prozesse aufeinander abgestimmt werden, um den Nutzen der Software für das Unternehmen zu maximieren.

Die Together-Funktionen sind in die Umgebung von RAD Studio integriert. Durch die Aktivierung der Together-Unterstützung werden folgende Oberflächenelemente hinzugefügt oder geändert:

- Diagrammansicht
- Modellansicht
- Objektinspektor
- Tool-Palette

Außerdem stehen im Hauptmenü und in den Kontextmenüs der Projektverwaltung und der Strukturansicht zusätzliche Befehle zur Verfügung.

Über die folgenden Quellen finden Sie weitere Unterstützung, Informationen und Ressourcen:

- Informationen darüber, wie Sie dieses Hilfesystem nutzen können, finden Sie im Thema Hilfe zur Hilfe (siehe Seite 1378).
- [Borland Together-Homepage](#)
- [Borland Together-Dokumentation](#)
- [Borland Produkt-Support](#)
- [Borland Newsgroups](#)

Warnung: Nicht alle in dieser Hilfe beschriebenen Funktionen stehen auch in jeder Edition des Produkts zur Verfügung.

Wenn Ihr Internet-Zugriff durch Netzwerksicherheit beschränkt oder Ihr Computer durch eine persönliche Firewall geschützt ist, kann es sein, dass die webbasierten Links in diesem Hilfesystem nicht korrekt funktionieren.

Siehe auch

[Überblick zur Modellierung](#) (siehe Seite 1446)

[Hilfe zur Hilfe \(siehe Seite 1378\)](#)

[Together Dokumentation](#)

[Together -Glossar \(siehe Seite 1266\)](#)

[Tastaturkürzel \(siehe Seite 1278\)](#)

7.8.2 Beispielprojekt "Cash Sales"

Die C#-Beispielanwendung "Cash Sales" zeigt die wichtigsten Aspekte eines einfachen Verkaufssystems.

Folgende Schritte werden ausgeführt:

7.8.2.1 Öffnen des C#-Beispielprojekts "Cash Sales"

So öffnen Sie das C#-Beispielprojekt "Cash Sales":

1. Klicken Sie im Menü **Datei** auf **Projekt öffnen**. Das Dialogfeld Projekt öffnen wird angezeigt.
2. Wechseln Sie zum Beispielverzeichnis der Together-Installation, z.B. C:\Programme\CodeGear\RAD Studio\5.0\Demos\CSCashSalesC:\Programme\Borland\Together VS .NET\Samples\CSCashSales.
3. Wählen Sie die Datei CashSales.csproj aus, und klicken Sie auf **Öffnen**.
4. Wählen Sie im Menü **Ansicht** den Befehl **Modellansicht**. Die Modellansicht wird zwar frei platzierbar geöffnet, sie kann aber angedockt werden. Sie können sie an die vier Ränder des RAD Studio-Fensters andocken.
5. Die Modellansicht kann beliebig positioniert werden. Erweitern Sie in der Modellansicht den Stammknoten des Projekts, und doppelklicken Sie auf das Diagramm **Open First**. Das Diagramm wird nun in der Diagrammansicht geöffnet.

In der Modellansicht befindet sich der Namespace **CSCashSales** in der obersten Ebene des Projekts. Von dort aus gelangen Sie zu den verschiedenen Namespaces und Diagrammen. Doppelklicken Sie in der Modellansicht auf die gewünschten Diagramme, um sie in der Diagrammansicht zu öffnen.

Siehe auch

[Beispielprojekt "Cash Sales" \(siehe Seite 1381\)](#)

[UML 2.0-Beispielprojekt \(siehe Seite 1384\)](#)

[Modellansicht \(siehe Seite 1270\)](#)

7.8.2.2 UML 1.5-Modellierung mit "Cash Sales"

In der Anwendung "Cash Sales" werden sämtliche UML-Diagramme verwendet. Mit Hilfe von Hyperlinks zu Elementen und Diagrammen können weitere Beziehungsebenen hinzugefügt werden.

So machen Sie sich mit einigen Funktionen der UML-Modellierung vertraut:

1. Erweitern Sie in der Modellansicht den Namespace-Knoten **CSCashSales**.
2. Klicken Sie mit der rechten Maustaste auf **POS**, und wählen Sie **Diagramm öffnen**. Sie können auch in der Modellansicht auf **POS** doppelklicken, um das Diagramm zu öffnen. Das Verteilungsdiagramm des Beispieldokuments wird nun in der Diagrammansicht geöffnet.

3. Erweitern Sie in der Modelansicht den Knoten **CSCashSales > requirements**. Doppelklicken Sie auf **Make a Sale**. Ein Anwendungsfalldiagramm wird geöffnet.
4. Klicken Sie in der Diagrammansicht mit der rechten Maustaste auf das Anwendungsfallelement **Scan Product**, und wählen Sie im Kontextmenü **Hyperlinks | CashSales.CSCashSales.requirements.Product Scanning Details**. Das Anwendungsfalldiagramm **Product Scanning Details** wird geöffnet.

Siehe auch

Beispielprojekt "Cash Sales" (siehe Seite 1381)

UML 2.0-Beispielprojekt (siehe Seite 1384)

Diagrammansicht (siehe Seite 1269)

7.8.2.3 Pattern in "Cash Sales" verwenden

Sie können den Quelltext des Projekts "Cash Sales" verwenden, um auf Basis der vorhandenen Diagrammelemente eigene Design-Pattern zu erstellen.

So erstellen Sie ein eigenes Design-Pattern:

1. Erweitern Sie in der Modellansicht den Knoten **CSCashSales > data_objects**.
2. Klicken Sie in der Modellansicht mit der rechten Maustaste auf die Klasse **CashSale**, und wählen Sie **Im Diagramm markieren**. Das Diagramm **data_objects** wird geöffnet und die Klasse hervorgehoben angezeigt.
3. Klicken Sie in der Diagrammansicht mit der rechten Maustaste auf die Klasse **CashSale**, und wählen Sie **Als Pattern speichern**. Der Experte zum Erstellen von Pattern wird geöffnet.
4. Legen Sie auf der ersten Seite des Experten folgende Einstellungen fest:
 - **Name:** Der Pattern-Name, der im Experten angezeigt wird.
 - **Datei:** Der Name der Pattern-Datei. Diese XML-Datei wird im Verzeichnis *..../Patterns/templets* der Together-Installation gespeichert.
 - **Beschreibung:** Geben Sie hier eine aussagekräftige Beschreibung des Pattern ein.
5. Klicken Sie auf **Weiter**. Die Seite **Pattern-Parameter** des Experten wird angezeigt.
6. Bearbeiten Sie ggf. die Parameter mit dem internen Editor, und klicken Sie auf **Weiter**. Die Seite **Hierarchieordner auswählen** des Experten wird mit der Struktur der Pattern-Hierarchie angezeigt.
7. Wählen Sie einen Zielordner für das Pattern aus, und klicken Sie auf **Fertig stellen**. Die neue Design-Template wird nun dem angegebenen Ordner hinzugefügt. Sie wird in der Pattern-Hierarchie angezeigt und kann zum Erstellen von Designelementen verwendet werden.

So verwenden Sie das neue Pattern:

1. Klicken Sie mit der rechten Maustaste auf den Diagrammhintergrund, und wählen Sie im Kontextmenü den Befehl **Nach Pattern erstellen**. Der Pattern-Experte wird geöffnet.
2. Wechseln Sie in der Pattern-Hierarchie zur neuen Template **CashSales**, und wählen Sie sie aus.
3. Ändern Sie im Bereich **Pattern-Eigenschaften** mit dem internen Editor den Klassennamen von **CashSale** in **NewCashSale**.
4. Klicken Sie auf **OK**, um eine neue Klasse zu erstellen. Die Klasse **NewCashSale** wird nun im Diagramm angezeigt.
5. Wählen Sie die Klasse **NewCashSale** aus, und drücken Sie die Taste ENTF, um sie aus dem Beispielprojekt zu entfernen.

Siehe auch

Beispielprojekt "Cash Sales" (siehe Seite 1381)

UML 2.0-Beispielprojekt (siehe Seite 1384)

7.8.2.4 Audits für "Cash Sales" ausführen

Sie können Audits für das Beispielprojekt "Cash Sales" ausführen. Das Ergebnis kann in einer Tabelle dargestellt, sortiert, kopiert und in eine HTML- oder XML-Datei exportiert werden.

So führen Sie Audits mit dem Beispielprojekt aus:

1. Wählen Sie im Hauptmenü **Tools** ▶ **QS-Audits**. Das Dialogfeld Audits wird geöffnet.
2. Wählen Sie im Listenfeld **Bereich** die Option **Modell** aus, um das gesamte Projekt zu überprüfen.
3. Wählen Sie die auszuführenden Audits aus. Sobald Sie auf ein Audit klicken, wird dessen Beschreibung im unteren Fenster des Dialogfelds angezeigt. Für jedes Audit werden der Schweregrad und die anderen audit-spezifischen Optionen auf der rechten Seite des Dialogfelds angezeigt.
4. Ändern Sie die Einstellungen bei Bedarf.
5. Wenn Sie die gewünschten Audits ausgewählt haben, klicken Sie auf **Start**. In der Statusleiste wird der Fortschritt der Operation angezeigt, bis sie abgeschlossen ist.

Die Audit-Ansicht wird automatisch mit dem Ergebnis angezeigt. Sie können in der Ergebnistabelle mit der rechten Maustaste auf die Zeilen klicken und über das Kontextmenü verschiedene Operationen mit dem Bericht durchführen.

Siehe auch

Beispielprojekt "Cash Sales" (siehe Seite 1381)

UML 2.0-Beispielprojekt (siehe Seite 1384)

Überblick über Qualitätssicherungsfunktionen (siehe Seite 1456)

Audits ausführen (siehe Seite 269)

7.8.2.5 Dokumentation für "Cash Sales" erzeugen

Sie können aus dem Beispielprojekt "Cash Sales" eine vollständige HTML-Dokumentation mit Navigationsleiste und Diagrammbildern erzeugen.

So generieren Sie die HTML-Dokumentation:

1. Klicken Sie in der Modellansicht mit der rechten Maustaste auf den Stammknoten des Projekts "Cash Sales", und wählen Sie **Dokumentation erzeugen**. Sie können auch im Hauptmenü **Tools** | **Dokumentation erzeugen** wählen. Das Dialogfeld **Dokumentation erzeugen** wird geöffnet.
2. Übernehmen Sie die Voreinstellungen im Dialogfeld **Dokumentation erzeugen**, und klicken Sie auf **OK**. Für das gesamte Beispielprojekt wird nun die Dokumentation erzeugt und in RAD Studio angezeigt.

Die Dokumentation wird im Browser in drei Frames angezeigt:

Dokumentations-Frames

Frame	Inhalt
Oberer Frame	Diagrammbilder
Unterer linker Frame	Projektbaum
Oberer rechter Frame	Textdokumentation

Die folgende Abbildung zeigt den unteren linken Frame der erzeugten Dokumentation. Die Bilder und der Text in der Dokumentation sind über Hyperlinks so verknüpft, wie es die Projektstruktur vorgibt. Wenn Sie in einem Bild auf einen

Namespace oder eine Klasse klicken, wird die entsprechende Textdokumentation geöffnet.

Siehe auch

Beispielprojekt "Cash Sales" ([siehe Seite 1381](#))

UML 2.0-Beispielprojekt ([siehe Seite 1384](#))

Überblick zur Dokumentationserzeugung ([siehe Seite 1458](#))

7.8.2.6 "Cash Sales" ausführen

So führen Sie das Beispielprojekt "Cash Sales" aus:

1. Klicken Sie im Menü **Debug** auf **Start**. Die Anwendung "Cash Sales" wird gestartet.
2. Um einen Kauf zu simulieren, klicken Sie auf **New Sale** und **Scan**. Beim Klicken auf die Schaltfläche **Scan** werden auf zufällige Weise Artikel aus der internen Verkaufsliste ausgewählt.
3. Klicken Sie auf **Total**, um die Gesamtsumme zu berechnen und die Schaltfläche **Payment** zu aktivieren. Wenn Sie hier einen Wert eingeben, berechnet die Anwendung das Wechselgeld oder gibt eine Warnung aus, falls die Zahlung nicht ausreicht.
4. Wenn Sie die Anwendung beenden möchten, klicken Sie auf **Close** oder im Menü **File** auf **Exit**.

Siehe auch

Beispielprojekt "Cash Sales" ([siehe Seite 1381](#))

UML 2.0-Beispielprojekt ([siehe Seite 1384](#))

7.8.3 UML 2.0-Beispielprojekt

Dieser Abschnitt enthält eine Beschreibung des UML 2.0-Beispielprojekts.

Folgende Schritte werden ausgeführt:

7.8.3.1 Das UML 2.0-Beispielprojekt öffnen

So öffnen Sie das UML 2.0-Beispielprojekt:

1. Klicken Sie im Menü **Datei** auf **Öffnen | Projekt**. Das Dialogfeld **Projekt öffnen** wird angezeigt.
2. Wechseln Sie zum Beispielverzeichnis der Together-Installation, z.B. C:\Programme\Borland\Together\Samples\UML-2.0.
3. Wählen Sie die Datei *UML-2.0.tgproj* aus, und klicken Sie auf **Öffnen**.
4. Wählen Sie im Menü **Ansicht** den Befehl Modellansicht. Die Modellansicht wird zwar frei platzierbar geöffnet, sie kann aber angedockt werden. Sie können sie an die vier Ränder des RAD Studio-Fensters andocken.
5. Das Fenster kann beliebig positioniert werden. Erweitern Sie in der Modellansicht den Stammknoten des Projekts, und doppelklicken Sie auf das Standarddiagramm. Das Diagramm wird nun in der Diagrammansicht geöffnet.

Doppelklicken Sie in der Modellansicht auf die gewünschten Diagramme, um sie in der Diagrammansicht zu öffnen.

Siehe auch

Beispielprojekt "Cash Sales" ([siehe Seite 1381](#))

UML 2.0-Beispielprojekt ([siehe Seite 1384](#))

7.8.3.2 UML 2.0-Beispielprojekt, Structure-Paket

Das Standarddiagramm auf der obersten Projektebene enthält zwei Pakete, die Ihnen die Struktur- und Verhaltensmodellierung veranschaulichen sollen.

Dieses Paket enthält die folgenden Diagramme:

- Klassendiagramm
- Komponentendiagramm
- Kompositionsstrukturdiagramm
- Verteilungsdiagramm

Klassendiagramm

Dieser Diagrammtyp wird durch drei Beispiele veranschaulicht: Class, Classes and Associations, Classes and Features.

Diagramm "Class"

Dieses Diagramm zeigt die Verwendung von Pattern als First-Class-Citizens (FCC). Dazu wird das GoF-Pattern "Abstract Factory" verwendet. Experimentieren Sie mit den verschiedenen Möglichkeiten zum Hinzufügen und Entfernen der Teilnehmer und Pattern-Objekte.

Diagramm "Classes and Associations"

Dieses Diagramm veranschaulicht folgende Funktionen:

- Generalisieren von Klassen
- Implementieren von Interfaces durch Klassen
- Arbeiten mit Assoziationsklassen und n-fachen Assoziationen. Beachten Sie, dass n-fache Assoziationen nur mit Hilfe einer Assoziationsklasse erstellt werden können.
- Arbeiten mit binären Assoziationen
- Bearbeiten der Eigenschaften von Assoziationsbeziehungen (gerichtete und ungerichtete Assoziationen, Beziehungsbeschriftungen, Client- und Anbieter-Multiplizität, Rollen, Qualifizierer, Einschränkungen usw.)

Diagramm "Classes and Features"

Dieses Diagramm veranschaulicht folgende Funktionen:

- Arbeiten mit Operationen und Attributen in den Klassen und Interfaces
- Bearbeiten der Eigenschaften von Operationen, Attributen und Slots (Sichtbarkeit, Multiplizität, Einschränkungen, Anfangswerte von Attributen und Slots, Rückgabetypen und Argumente von Operationen)
- Instantiieren von Klassen durch Instanzspezifikationen
- Definieren von Funktionsweisen durch Slots

Komponentendiagramm

Das Beispieldiagramm "Store Components" zeigt Folgendes:

- Arbeiten mit inneren Komponenten
- Verwenden erforderlicher und bereitgestellter Interfaces über Ports
- Delegieren der Aufrufe einer Komponente durch Delegationskonnektoren an ihre Unterkomponenten

Kompositionssstrukturdiagramm

Dieses Beispieldiagramm zeigt die Verwendung von Kollaborationen und Parts.

Verteilungsdiagramm

Das Beispieldiagramm "Application Server" zeigt die Verwendung von Verteilungsspezifikationen und Artefakten.

Siehe auch

Beispielprojekt "Cash Sales" (siehe Seite 1381)

UML 2.0-Beispielprojekt (siehe Seite 1384)

Überblick zur Modellierung (siehe Seite 1446)

7.8.3.3 UML 2.0-Beispielprojekt, Behavior-Paket

Dieses Paket enthält folgende Diagramme:

- Aktivitätsdiagramm
- Zustandsmaschinendiagramm
- Anwendungsfalldiagramm
- Interaktionsdiagramm

Aktivitätsdiagramm

Dieser Diagrammtyp wird durch drei Beispiele veranschaulicht:

- Data Activity
- Final Nodes
- Process Order

Diagramm "Data Activity"

Dieses Diagramm zeigt den Objektfluss über Aktionen und Pins sowie die Verwendung des zentralen Puffers.

Diagramm "Final Nodes"

Dieses Diagramm bildet das Erstellen einer Anwendung mit mehreren Kontrollabläufen und Terminal-Blöcken nach. Während dieses Vorgangs werden die Anwendungskomponenten zusammengestellt. Danach wird der Erstellungsablauf abgeschlossen (Ablaufende) und der Installationsablauf gestartet. Nachdem alle Komponenten erstellt und installiert sind, wird die Aktion "Deliver" ausgeführt und dadurch die gesamte Aktivität abgeschlossen (Aktivitätsende).

Diagramm "Process Order"

Dieses Diagramm zeigt die Interaktionen zwischen den verschiedenen Aktionen durch Kontrollabläufe, die Übertragung von Informationen durch Objektflüsse und die Verwendung der Signalsendungs- und Signalempfangselemente.

Zustandsmaschinendiagramm

Dieser Diagrammtyp wird durch zwei Beispiele veranschaulicht:

- Course Attempt
- Submachine State

Diagramm "Course Attempt"

Dieses Diagramm zeigt die Verwendung von Unterzuständen und Regionen. Da Zustände keine untergeordneten Elemente desselben Typs enthalten können, werden die verschachtelten Unterzustände in die Regionen eingefügt.

Diagramm "Submachine State"

Dieses Diagramm zeigt, wie in einem Zustand auf ein anderes Diagramm verwiesen werden kann. "ReadAmountSM" ist ein Zustandelement, das einem gesamten Diagramm entspricht, und "ReadAmount:ReadAmountSM" ist ein Zustand, der das Verhalten von "ReadAmountSM" implementiert. Beide Zustände sind durch Hyperlinks verknüpft.

Anwendungsfalldiagramm

Das Beispieldiagramm "Main Use Cases" zeigt die Verwendung von Subjekten und Stereotypen.

Interaktionsdiagramm

Interaktionsdiagramme zeigen die Interaktion zwischen Objekten (Lebenslinien) durch Nachrichten. Jede Lebenslinie instantiiert eine Klasse oder entspricht einem Part. Eine Interaktion kann auf zwei Arten dargestellt werden: als Sequenzdiagramm oder als Kommunikationsdiagramm. Die Beispielinteraktion "ShowAlbumsDialog" wird durch das Diagramm "ShowAlbumsDialog" (Sequenz) und das Diagramm "!cd_ShowAlbumsDialog" (Kommunikation) modelliert.

Siehe auch

Beispielprojekt "Cash Sales" ([siehe Seite 1381](#))

UML 2.0-Beispielprojekt ([siehe Seite 1384](#))

Überblick zur Modellierung ([siehe Seite 1446](#))

8 Anleitungen

Dieser Abschnitt enthält Anleitungen zu verschiedenen Bereichen der Entwicklung mit RAD Studio.

8.1 Einführende Anleitungen

Dieser Abschnitt enthält unter anderem Anleitungen zur Konfiguration der IDE sowie zur Arbeit mit Formularen und Projekten.

8.2 Compilieren und Builds erstellen – Anleitungen

Dieser Abschnitt enthält Anleitungen zum Erstellen von Packages und zum Lokalisieren von Anwendungen.

8.3 Debugging – Anleitungen

Dieser Abschnitt enthält Anleitungen zur Fehlersuche in Anwendungen.

8.4 Quelltext bearbeiten – Anleitungen

Dieser Abschnitt enthält Anleitungen zur Verwendung des Quelltext-Editors.

8.5 Lokalisierung von Anwendungen – Anleitungen

Anleitungen zur Lokalisierung von Anwendungen mit den Übersetzungs-Tools von RAD Studio.

8.6 Together-Diagramme – Anleitungen

Dieser Abschnitt enthält Anleitungen zur Verwendung von Together-UML-Diagrammen.

8.7 Together-Dokumentationserzeugung – Anleitungen

Dieser Abschnitt enthält Anleitungen zur Verwendung der Funktionen zur Dokumentationserzeugung in Together.

8.8 Together Object Constraint Language (OCL) – Anleitungen

Dieser Abschnitt enthält Anleitungen zur Verwendung der OCL-Funktionen von Together.

8.9 Together-Pattern – Anleitungen

Dieser Abschnitt enthält Anleitungen zur Verwendung von Pattern mit Together.

8.10 Together-Projekte – Anleitungen

Dieser Abschnitt enthält Anleitungen zur Verwendung von Together-Projekten.

8.11 Together-Qualitätssicherung – Anleitungen

Dieser Abschnitt enthält Anleitungen zur Verwendung der Qualitätssicherungsfunktionen von Together.

8.12 Together-Refactoring – Anleitungen

Dieser Abschnitt enthält Anleitungen zur Verwendung der Refactoring-Funktionen von Together.

8.13 Testen von Units – Anleitungen

Anleitungen zur Verwendung der Funktionen der Test-Frameworks DUnit and NUnit.

9 Templates (Fenster)

Ansicht ▾ Templates

Verwenden Sie das Fenster **Templates** zum Erstellen, Bearbeiten oder Löschen von Quelltext-Templates.

Element	Beschreibung
Name	Zeigt die Namen der verfügbaren Quelltext-Templates an.
Beschreibung	Beschreibt die Quelltext-Template.

Symbolleiste des Fensters *Template*

Symbolleistenschaltfläche	Beschreibung
Neue Quelltext-Template	Erstellt im Quelltext-Editor eine XML-Template-Datei mit Standardcode, den Sie bearbeiten können. Legen Sie den Template-Namen, den Autor, die Beschreibung und den Inhalt der Quelltext-Template fest.
Quelltext-Template entfernen	Entfernt die ausgewählte Quelltext-Template aus der Liste und löscht die <code>.xml</code> -Datei der Template vom Datenträger.
Quelltext-Template bearbeiten	Öffnet die ausgewählte Template im Quelltext-Editor, in dem Sie sie bearbeiten können.
Quelltext-Template in den Quelltext-Editor einfügen	Fügt den Inhalt aus der ausgewählten Template an der Cursorposition in den Quelltext-Editor ein.
Quelltext-Template nach Sprache filtern	Zeigt nur die Templates an, die sich auf die aktuelle Projektsprache beziehen.

Siehe auch

Quelltext-Templates erstellen (↗ siehe Seite 43)

10 Entwicklungszyklus steuern

Die Anwendungsentwicklung besteht aus dem Entwerfen, Programmieren, Testen, Debuggen und Weitergeben von Anwendungen. In RAD Studio stehen leistungsstarke Tools zur Auswahl, die diesen Prozess unterstützen. Dazu gehören die Formularentwurfs-Tools, der Delphi-Compiler, eine integrierte Debugging-Umgebung sowie Tools zur Installation und zum Deployment.

10.1 Überblick zum Steuern des Entwicklungszyklus

Der hier beschriebene Entwicklungszyklus ist ein Teil des Application Lifecycle Management (ALM) und betrifft speziell jenen Bereich des Zyklus, der die Implementierung und Steuerung der eigentlichen Entwicklungsaufgaben umfasst. Der hier beschriebene Entwicklungszyklus enthält keine Erläuterung der Anwendungsmodellierung.

Zu den ALM-Tools zählen:

- Anforderungsverwaltung
- SourceControl
- Entwurf von Benutzeroberflächen
- Möglichkeiten der Code-Visualisierung
- Erstellen und Compilieren von Projekten sowie Debug-Möglichkeiten

Anforderungsverwaltung

Mit den Tools für die Anforderungsverwaltung können Sie Anforderungen für Ihr Software-Projekt hinzufügen, entfernen und aktualisieren. Außerdem können Sie Beziehungen zwischen der Anforderungsspezifikation und den Abschnitten im Quelltext erstellen, die die Anforderung erfüllen.

SourceControl

Mit einem SourceControl-System Verwalten Sie verschiedene Versionen Ihrer Projektdateien verwalten. Die meisten SourceControl-Systeme verfügen über ein zentrales Repository für den Quelltext und ermöglichen das Einchecken, Auschecken, Aktualisieren, Übergeben sowie andere Verwaltungsaufgaben für Quelltextdateien.

Entwurf von Benutzeroberflächen

RAD Studio ist eine leistungsstarke Umgebung für den Entwurf von .NET-Benutzeroberflächen. Neben dem Windows Forms-Designer, der eine Sammlung visueller Komponenten enthält, stellt die IDE Tools zum Erstellen von ASP.NET-Web Forms sowie eine Sammlung von Web Controls zur Verfügung.

Die IDE enthält ferner ein Entwurfs-Tool für VCL.NET-Formulare, mit dessen Hilfe Sie .NET-Anwendungen mit den VCL-Komponenten erstellen können. Der Designer bietet viele Tools für die Ausrichtung, Schriften und visuelle Komponenten, um Sie beim Erstellen der verschiedensten Anwendungsarten zu unterstützen, zum Beispiel MDI- und SDI-Anwendungen, Dialogfelder mit Registern und datensensitive Anwendungen.

Code-Visualisierung

Die Funktion der Code-Visualisierung von RAD Studio bietet Instrumente zur Dokumentation und zum Debuggen von Klassenentwürfen anhand eines visuellen Musterbeispiels. Wenn Sie Ihre Projekte und Quelltextdateien laden, können Sie die Modellansicht dazu nutzen, sowohl eine hierarchisch strukturierte Ansicht aller in den Klassen repräsentierten Objekte, als auch ein UML-ähnliches Modell der Anwendungsobjekte anzuzeigen. Diese Funktion visualisiert die Beziehungen zwischen den Objekten in Ihrer Anwendung und unterstützt Sie dadurch bei der Entwicklung und Implementierung.

Compilieren, Erstellen, Ausführen und Debuggen

RAD Studio enthält MSBuild, die Build-Engine auf Industriestandard, zusammen mit einem integrierten Debugger. Mit dem Befehl Compilieren erstellen Sie nur die geänderten Elemente des Projekts. Um das gesamte Projekt unabhängig von Änderungen zu erstellen, verwenden Sie den Befehl Erzeugen. Projekte mit Unterprojekten und mehreren Quelltextdateien können zusammen oder einzeln erstellt werden.

Der integrierte Debugger bietet die Möglichkeit, überwachte Ausdrücke und Haltepunkte zu setzen und bestimmte Codezeilen im Einzelschritt zu überprüfen oder auszulassen. Die verschiedenen Fenster des Debuggers zeigen Detailinformationen zu Variablen, Prozessen und Threads an. Dadurch haben Sie die Möglichkeit, den Code bis ins Detail zu prüfen, um nach Fehlern zu suchen und diese zu beheben.

Siehe auch

Compilieren ( [siehe Seite 1351](#))

Packages erstellen ( [siehe Seite 3](#))

Anwendungen debuggen ( [siehe Seite 1439](#))

10.2 Mit der SourceControl arbeiten

Dieses Thema gibt einen Überblick über die allgemeinen Konzepte der Versionskontrolle, die für eine Reihe von Versionskontrollsysten gelten, die auch unter der Bezeichnung "Automatisiertes System zur Verwaltung von Änderungen und Softwarekonfigurationen (SCM)" bekannt sind. .

Grundlagen der SourceControl

Jedes SourceControl-System besteht aus einem oder mehreren zentralen Repositories und einigen Clients. Ein Repository ist eine Art Archiv in einer Datenbank, das nicht nur die aktuellen Datendateien enthält, sondern auch die Struktur der einzelnen Projekte aufzeichnet.

Die meisten SourceControl-Systeme bedienen sich des Konzepts eines logischen *Projekts*, d.h. eines Projekts, in dem die Dateien in einer oder mehreren Verzeichnisstrukturen gespeichert sind. Ein SourceControl-System kann ein oder mehrere RAD Studio-Projekte sowie weitere Dokumente und Zubehördateien enthalten. Diese Systeme sehen gelegentlich eine eigene Benutzeroauthentifizierung vor, verwenden aber häufig auch die bereits im Betriebssystem angelegte Authentifizierungsmethode. Dadurch wird es dem SourceControl-System möglich, ein Protokoll oder eine Momentaufnahme der Aktualisierung jeder einzelnen Datei festzuhalten. Diese Momentaufnahmen werden häufig als *Diffs* (Differenzen) bezeichnet. Wenn nur die Differenzen gespeichert werden, kann das SourceControl-System alle Änderungen mit nur minimalen Speicheranforderungen protokollieren. Sobald Sie eine komplette Kopie einer dieser Dateien ansehen möchten, führt das System eine Zusammenführung der Differenzen durch und zeigt Ihnen das komplette Ergebnis an. Diese Differenzen werden zunächst in verschiedenen Dateien gespeichert, bis Sie eine Zusammenführung der Aktualisierungen veranlassen. Dazu können Sie die Aktion *Übernehmen* ausführen.

Dieser Ansatz ermöglicht es Ihnen und anderen Teammitgliedern, parallel und gleichzeitig an mehreren gemeinsamen Projekten zu arbeiten, ohne dass der Quelltext eines Mitglieds von einem anderen überschrieben werden kann. Die SourceControl-Systeme verhindern im Wesentlichen Code-Konflikte und einen eventuellen Verlust älteren Quelltextes. Die meisten SourceControl-Systeme enthalten Tools zur Verwaltung der Quelltextdateien mit der Möglichkeit des Ein- oder Auscheckens von Dateien, zur Konfliktlösung und zur Berichtserstellung. Funktionen zur logischen Konfliktlösung oder zum Build-Management sind jedoch in den meisten Systemen nicht vorgesehen. Genauere Informationen zu den einzelnen SourceControl-Systemen finden Sie in den Dokumentationen zu den Produkten der verschiedenen Hersteller.

Im Allgemeinen bieten SourceControl-Systeme nur die Möglichkeit, verschiedene Versionen textbasierter Dateien (z.B. Quelltext-, HTML- und XML-Dateien) zu vergleichen und zusammenzuführen. Bei einigen SourceControl-Systemen können zusätzlich Binärdateien in die Projekte in der SourceControl eingefügt werden. Sie können die Versionen der Binärdateien jedoch nicht vergleichen oder zusammenführen. Wenn Sie nicht nur bestimmte Versionen dieser Dateien speichern und abrufen möchten, sollten Sie ein manuelles Protokollsystem für Änderungen in Binärdateien in Betracht ziehen.

Grundlagen des Repository

SourceControl-Systeme speichern die Kopien von Quelltextdateien und Differenzdateien in einem Datenbankarchiv, das als Repository bezeichnet wird. In einigen Systemen, wie zum Beispiel CVS oder VSS, ist das Repository eine logische Struktur, die aus linearen Dateien und Steuerungsdateien besteht. In anderen Systemen sind die Repositories Instanzen eines bestimmten Datenbank-Management-Systems (DBMS), zum Beispiel von InterBase, Microsoft Access, MS SQL Server, IBM DB2 oder Oracle.

Die Repositories sind üblicherweise auf einem externen Server gespeichert, damit mehrere Benutzer gleichzeitig eine Verbindung dazu herstellen, Dateien ein- oder auschecken und andere Management-Aufgaben ausführen können. Sie müssen sicherstellen, dass Ihre Verbindung nicht nur zum Server, sondern auch zu der betreffenden Datenbankinstanz besteht. Halten Sie mit den zuständigen Netzwerk-, System- oder Datenbankverwaltern Rücksprache, um sicherzustellen, dass Ihr Computer mit den erforderlichen Treibern und der notwendigen Verbindungssoftware sowie der SourceControl-Software für den Client

ausgestattet ist.

Bei einigen SourceControl-Systemen lässt sich ein lokales Repository erstellen, in dem Sie die Momentaufnahme Ihrer Projekte auf Ihrem System aufbewahren können. Mit zunehmender Bearbeitung wird sich das lokale Abbild Ihrer Projekte vom Inhalt des Repositories auf dem externen Server immer mehr unterscheiden. Sie können einen regelmäßigen Austausch zum Zusammenführen und für die Übergabe Ihrer Änderungen vom lokalen Repository an das externe Repository durchführen.

Im Allgemeinen ist es jedoch nicht zu empfehlen, jedes Mitglied eines Teams mit einem eigenen Repository für ein gemeinsames Projekt auszustatten. Wenn Sie jedoch an ganz getrennten Projekten arbeiten, und die SourceControl für alle diese Projekte lokal an einem Computer durchgeführt werden soll, kann es sinnvoll sein, mit einzelnen lokalen Repositories zu arbeiten. Sie können auch mehrere Repositories auf einem externen Server einrichten, wodurch eine zentralisierte Unterstützung, Sicherung und Verwaltung gewährleistet ist.

Mit Projekten arbeiten

Die SourceControl-Systeme verwenden, ähnlich wie die Entwicklungsumgebungen, das Konzept des Projekts, um Gruppen zusammengehörender Dateien zu verwalten bzw. deren Status zu protokollieren. In allen SourceControl-Systemen wird ein Projekt erstellt, das die Dateidefinitionen und die zugehörigen Speicherpositionen enthält. Auch in RAD Studio erstellen Sie Projekte, um die verschiedenen Assemblierungen und Quellcode-Dateien für eine Anwendung zu verwalten. In RAD Studio werden die Parameter eines Projekts in einer Projektdatei gespeichert. Sie können auch diese Datei zusätzlich zu den verschiedenen Quelltextdateien im SourceControl-System speichern. Dies ist sinnvoll, wenn Sie die Projektdatei gemeinsam mit anderen Programmierern im Team verwenden möchten oder wenn jeder eine eigene Projektdatei aufbewahren soll.

Die meisten SourceControl-Systeme interpretieren die Projektdateien aus den Entwicklungsumgebungen als Binärdatei und zwar unabhängig davon, ob sie es tatsächlich sind oder nicht. Daher werden beim Einchecken einer Projektdatei in das Repository eines SourceControl-Systems ältere Versionen der Datei durch neuere Versionen überschrieben, ohne dass eine Zusammenführung der Änderungen versucht wird. Dies gilt auch für das Abrufen eines Projekts oder Auschecken der Projektdatei. Die ältere Version wird ohne Zusammenführung durch die neuere Dateiversion überschrieben.

Mit Dateien arbeiten

In der Objekthierarchie stellen Dateien die unterste Ebene dar, die sich in einem SourceControl-System verwalten lässt. Jeder Quelltext, der in der SourceControl überwacht werden soll, muss in einer Datei enthalten sein. Die meisten SourceControl-Systeme speichern Dateien in Form einer logischen, hierarchischen Struktur. In einigen Systemen, zum Beispiel CVS, werden Begriffe wie *Verzweigung* verwendet, um Verzeichnisebenen zu benennen. Sie können Dateien in einem RAD Studio-Projekt erstellen und diese dann in das SourceControl-Projekt einfügen oder bereits vorhandene Dateien aus dem SourceControl-Projekt übernehmen. Ein komplettes Verzeichnis lässt sich ebenfalls in das SourceControl-System einfügen. Später können Sie dann einzelne Dateien oder Unterverzeichnisse daraus wieder auschecken. In RAD Studio haben Sie Ihre Dateien auf zwei Ebenen unter Kontrolle: auf der Projektebene innerhalb von RAD Studio und durch die Schnittstelle zwischen RAD Studio und dem SourceControl-System auch im jeweiligen SourceControl-System selbst.

Anmerkung: Im Fenster Versionsgeschichte werden Versionsinformationen zu Ihren lokalen Quelltextdateien angezeigt. Sie können das Fenster Versionsgeschichte verwenden, um während der Bearbeitung der Dateien im Designer oder Quelltext-Editor die Änderungen zu überprüfen.

Siehe auch

Mit der Versionsverwaltung arbeiten (siehe Seite 52)

10.3 Benutzeroberflächen entwerfen

Eine grafische Benutzeroberfläche (GUI) besteht aus einem oder mehreren Fenstern, die es dem Benutzer ermöglichen, mit einer Anwendung in Interaktion zu treten. Zur Entwurfszeit werden diese Fenster als *Formular* oder kurz *Form* bezeichnet. In RAD Studio steht für die Erstellung von Windows Forms, Web Forms und HTML-Seiten ein Designer zur Verfügung. Die Designer und Formulare unterstützen Sie dabei, professionelle Benutzeroberflächen schnell und einfach zu erstellen.

Mit dem Designer arbeiten

Wenn Sie eine Windows-, Web- oder Web-Services-Anwendung erstellen, zeigt die IDE automatisch den entsprechenden Formulartyp auf der Registerkarte Design in der IDE an. Sobald Sie Komponenten wie Labels und Textfelder aus der Tool-Palette in das Formular ziehen, wird der zugrunde liegende Quelltext für die Funktionalität der Anwendung automatisch in RAD Studio erstellt. Die Eigenschaften der Komponenten oder des Formulars ändern Sie im Objektinspektor. Die Ergebnisse dieser Änderungen werden automatisch in den Quelltext auf dem Register Quelle übernommen. Umgekehrt werden Änderungen am Quelltext, die im Quelltext-Editor durchgeführt wurden, sofort in das Register Design übernommen.

In der Tool-Palette stehen Dutzende von Steuerelementen zur Auswahl, mit deren Hilfe der Entwurf von Windows Forms, Web Forms und HTML-Seiten vereinfacht wird. Wenn Sie ein Windows Form erstellen, können Sie beispielsweise die Komponente MainMenu dazu verwenden, in nur wenigen Minuten ein Hauptmenü zu entwerfen. Wenn sich die Komponente auf einem Windows Form befindet, geben Sie die Hauptmenüinträge in die dafür vorgesehenen Felder ein. Die Komponente ContextMenu enthält eine ähnliche Funktionalität zum schnellen Erstellen von Kontextmenüs. Es gibt ferner einige Dialogfeld-Komponenten für allgemein übliche Funktionen, wie zum Beispiel das Öffnen und Speichern von Dateien, das Einstellen von Schriftarten und Farben sowie das Drucken. Wenn Sie diese Komponenten verwenden, sparen Sie auf der einen Seite Zeit und auf der anderen Seite sorgen Sie für ein konsistentes Look-and-Feel der Dialoge in Ihren Anwendungen.

Beim Entwurf einer Benutzeroberfläche können Sie Eingaben rückgängig machen oder diesen Schritt widerrufen, indem Sie **Bearbeiten**▶**Rückgängig** oder **Bearbeiten**▶**Wiederherstellen** wählen. Wenn Sie mit dem Erscheinungsbild des Formulars zufrieden sind, können Sie die darauf positionierten Komponenten sperren, um unbeabsichtigte Veränderungen zu verhindern. Dazu klicken Sie das Formular mit der rechten Maustaste an und wählen aus dem Kontextmenü Elemente fixieren.

Designer-Optionen einstellen

Sie können Optionen einstellen, die das Erscheinungsbild und das Verhalten der Designer ändern. Zum Beispiel lassen sich die Rastereinstellungen und der Stil des generierten Quelltextes oder des HTML-Codes anpassen. Um diese Optionen einzustellen, wählen Sie **Tools**▶**Optionen** und verwenden dann die Dialogfelder Windows Forms-Designer und HTML-Designer-Optionen.

Designer-Richtlinien für VCL-Komponenten definieren

Mit VCL und VCL.NET (Delphi oder C++) können Sie Komponenten so definieren, dass sie sich "relativ" zu den anderen Komponenten im Formular verhalten. Beispielsweise können Sie für eine Komponente veranlassen, dass abhängig vom Wert der Eigenschaft *padding* immer ein bestimmter Freiraum um die Komponente erhalten bleibt.

Mit Hilfe entsprechender Eigenschaften lässt sich der Abstand zwischen Steuerelementen, Verknüpfungen, Fokusbeschriftungen, der Tabulatorreihenfolge und den Einträgen von Listenfeldern und Menüs festlegen.

Nachdem Richtlinien für Komponenten definiert wurden, können diese bei aktiverter Option **Designer-Richtlinien verwenden** zur Erstellung von Formularen (und zum Generieren entsprechenden Quelltextes) verwendet werden. Wenn sowohl die Option **Am Raster ausrichten** als auch die Verwendung von Designer-Richtlinien aktiviert ist, haben die Designer-Richtlinien Vorrang.

Anleitungen zum Festlegen von Designer-Richtlinien finden Sie über den entsprechenden Link am Ende dieses Themas.

Siehe auch

Überblick zu Windows Forms

Übersicht zu ASP.NET

Komponenten in ein Formular einfügen (siehe Seite 55)

Designer-Richtlinien für VCL-Komponenten verwenden (siehe Seite 80)

Windows Forms-Menüs erstellen

10.4 Refactoring von Anwendungen

Refactoring ist ein Verfahren zum Umstrukturieren und Bearbeiten des Quelltextes, ohne dessen Funktionalität zu ändern. RAD Studio stellt Refactoring-Funktionen bereit, mit denen Sie den Quelltext Ihrer Anwendungen vereinheitlichen und vereinfachen können. Der Quelltext wird dadurch besser lesbar und kann leichter gepflegt werden.

10.4.1 Überblick zum Refactoring

Refactoring ist ein Verfahren zum Umstrukturieren und Bearbeiten des vorhandenen Quelltextes, ohne dessen Funktionalität zu ändern. Mittels Refactoring können Sie den Quelltext Ihrer Anwendungen vereinheitlichen und vereinfachen. Der Quelltext wird dadurch besser lesbar und kann leichter gepflegt werden.

Eine Refactoring-Operation betrifft immer einen bestimmten Typ von Bezeichner. Durch eine Folge von Refactoring-Operationen lässt sich die Quelltextstruktur wesentlich verändern. Da jede Operation aber nur einen bestimmten Objekt- oder Aktionstyp betrifft, ist die Fehlerquote gering. Sollte eine Operation zu einem unerwünschten Ergebnis führen, kann sie jederzeit rückgängig gemacht werden. Für jede Refactoring-Operation gibt es Einschränkungen, die beachten werden müssen. So ist es beispielsweise nicht möglich, vom Compiler importierte Symbole umzubenennen. In den Themen zu den einzelnen Refactoring-Funktionen wird auf diese speziellen Einschränkungen hingewiesen.

Zu RAD Studio gehört eine Refactoring-Engine, die Auswertungen durchführt und die Refactoring-Operation ausführt. Die Engine zeigt die Auswirkungen der Refactoring-Operation in einem Vorschaufenster am unteren Rand des Quelltext-Editors an. Die möglichen Refactoring-Operationen werden im Baumdiagramm als Knoten dargestellt. Sie können erweitert werden, um zusätzliche Elemente anzuzeigen, die eventuell vom Refactoring betroffen sind. Im Vorschaufenster werden auch Warn- und Fehlermeldungen angezeigt. Der Zugriff auf die Refactoring-Funktionen erfolgt über das Hauptmenü und über kontextbezogene Dropdown-Menüs.

RAD Studio unterstützt die folgenden Refactoring-Operationen:

- Symbol umbenennen (Delphi, C#, C++)
- Methode extrahieren (Delphi)
- Variable und Feld deklarieren (Delphi)
- Sync-Bearbeitungsmodus (Delphi, C#)
- Referenzen suchen (Delphi, C#, C++)
- Ressourcenstring extrahieren (Delphi)
- Unit suchen (Delphi)
- Namespace verwenden (C#)
- Rückgängig (Delphi, C#)
- Parameter ändern (Delphi)

Siehe auch

Überblick zum Umbenennen von Symbolen (siehe Seite 1413)

Refactoring von Quelltext (siehe Seite 1415)

Refactoring-Operationen in der Vorschau anzeigen und durchführen (siehe Seite 1417)

Sync-Bearbeitungsmodus (siehe Seite 1422)

Überblick zum Extrahieren von Methoden (siehe Seite 1414)

Überblick zum Suchen von Referenzen (siehe Seite 1421)

Überblick zum Deklarieren von Variablen und Feldern (siehe Seite 1419)

Überblick zum Extrahieren von Ressourcenstrings (siehe Seite 1417)

Referenzen suchen (siehe Seite 1423)

Umbenennen rückgängig machen (siehe Seite 1422)

Units suchen und Namespaces verwenden (siehe Seite 1424)

10.4.2 Überblick zum Umbenennen von Symbolen (Delphi, C#, C++)

Die Refactoring-Engine unterstützt das Umbenennen von Bezeichnern und aller Referenzen auf den Zielbezeichner. Sie können einen Bezeichner umbenennen, wenn sich der ursprüngliche Deklarationsbezeichner innerhalb Ihres Projekts oder in einem Projekt befindet, von dem ihr Projekt abhängig ist (in der Projektgruppe). Es ist auch möglich, Fehlerbezeichner umzubenennen (z.B. einen nicht deklarierten Bezeichner oder Typ).

Folgende Regeln sind beim Umbenennen von Symbolen mit der Refactoring-Engine zu beachten:

- Ein Bezeichner kann nicht in ein Schlüsselwort umbenannt werden.
- Ein Bezeichner kann nur in denselben Bezeichner umbenannt werden, wenn sich die Groß-/Kleinschreibung unterscheidet.
- Ein Bezeichner kann in einem abhängigen Projekt nur dann umbenannt werden, wenn das Projekt mit dem ursprünglichen Deklarationsbezeichner geöffnet ist.
- Vom Compiler importierte Symbole können nicht umbenannt werden.
- Eine überschriebene Methode, deren Basismethode in einer Klasse deklariert ist, die nicht Teil des Projekts ist, kann nicht umbenannt werden.
- Wenn beim Refactoring ein Fehler auftritt, kann die Engine die Änderung nicht durchführen. Es ist zum Beispiel nicht möglich, einem Bezeichner einen Namen zuzuweisen, der in demselben Deklarationsbereich bereits vergeben ist. Soll der Bezeichner trotzdem diesen Namen erhalten, müssen Sie vorher den Bezeichner umbenennen, das gegenwärtig diesen Namen hat, und danach das Refactoring aktualisieren. Sie können das Refactoring auch rückgängig machen und einen neuen Namen wählen. Die Refactoring-Engine sucht über Bereichsgrenzen hinweg nach einem Bezeichner mit demselben Namen. Wird ein Bezeichner mit diesem Namen gefunden, gibt die Engine eine Warnung aus.

Methoden umbenennen

Das Umbenennen von Methoden, Typen oder anderen Objekten erfolgt auf die gleiche Weise wie bei einem Bezeichner. Sie können im Quelltext-Editor einen Prozedurnamen auswählen und ändern. Ist die Prozedur überladen, benennt die Refactoring-Engine nur die überladene Prozedur um und ruft nur diese auf: Das folgende Codefragment veranschaulicht diese Regel:

```
procedure Foo; overload;
procedure Foo(A:Integer); overload;
Foo();
Foo;
Foo(5);
```

Wenn Sie den Namen der ersten *Foo*-Prozedur in diesem Beispiel ändern, benennt die Engine das erste, dritte und vierte Element um.

Wenn Sie den Namen eines überschriebenen Bezeichners ändern, benennt die Engine alle Basisdeklarationen und alle abgeleiteten Deklarationen um, d.h. der ursprüngliche virtuelle Bezeichner sowie alle vorhandenen überschriebenen Symbole. Das folgende Codefragment veranschaulicht diese Regel:

```
TFoo = class
    procedure Foo; virtual;
end;

TFoo2 = class(TFoo)
    procedure Foo; override;
end;

TFoo3 = class(TFoo)
    procedure Foo; override;
end;

TFoo4 = class(TFoo3)
    procedure Foo; override;
end;
```

Wenn Sie *Foo* umbenennen, werden alle Instanzen von *Foo* umbenannt.

Siehe auch

Überblick zum Refactoring (siehe Seite 1412)

Überblick zum Extrahieren von Methoden (siehe Seite 1414)

Ein Symbol umbenennen (siehe Seite 11)

10.4.3 Überblick zum Extrahieren von Methoden (Delphi)

Mit dem Refactoring-Befehl Methode extrahieren kann ein Codefragment in eine Methode konvertiert werden, deren Name den Zweck der Methode beschreibt. Der zuvor ausgewählte Quelltext wird dazu analysiert. Kann dieser Quelltext nicht zu einer Methode extrahiert werden, gibt die Refactoring-Engine eine Warnung aus. Ist ein Refactoring möglich, erzeugt die Engine außerhalb der aktuellen Methode eine neue Methode. Anschließend definiert die Engine die erforderlichen Parameter, generiert lokale Variablen, bestimmt den Rückgabetyp und fordert den Benutzer zu Eingabe eines Namens auf. Die Refactoring-Engine fügt einen Aufruf der neuen Methode an der Position der alten Methode ein.

Beim Extrahieren von Methoden sind einige Einschränkungen zu beachten. Dies sind:

- Es können nur Anweisungen, aber keine Ausdrücke extrahiert werden.
- In Delphi können keine Anweisungen extrahiert werden, die einen Aufruf von **inherited** enthalten.
- Es können keine Anweisungen innerhalb eines **with**-Blocks extrahiert werden.
- Anweisungen, die Aufrufe einer lokalen Prozedur oder Funktion enthalten, können nicht extrahiert werden.

Wenn Sie einen Ausdruck auswählen und den Befehl Methode extrahieren aufrufen, wird die Auswahl auf die gesamte Anweisung erweitert. Dient der Ausdruck in der Anweisung als Ergebnis, gibt der extrahierte Quelltext anstelle des Ausdrucks ein Funktionsergebnis zurück.

Siehe auch

Überblick zum Refactoring (siehe Seite 1412)

Refactoring von Quelltext (siehe Seite 1415)

Ein Symbol umbenennen (siehe Seite 11)

10.4.4 Refactoring von Quelltext

Unter Refactoring versteht man die Durchführung struktureller Änderungen am Programmcode, ohne dessen Funktionalität zu ändern. Durch das Refactoring wird der Code kompakter, besser lesbar und effizienter. RAD Studio unterstützt verschiedene Refactoring-Operationen, mit denen Sie die Struktur Ihres Quelltextes verbessern und optimieren können.

Refactoring-Operationen können für Delphi- und C#- und C++ Quelltext durchgeführt werden. Die Zahl der Refactoring-Funktionen für C# und C++ ist jedoch begrenzt. Der Zugriff auf die Refactoring-Befehle kann entweder über das Menü Refactoring oder über ein Kontextmenü im Quelltext-Editor erfolgen.

Alle Refactoring-Operationen können rückgängig gemacht werden. Bei vielen Operationen erfolgt dies mit dem Standard-Menübefehl Rückgängig (STRG+Z). Die Refactoring-Funktion zum Umbenennen verfügt über einen eigenen Rückgängig-Mechanismus.

So benennen Sie ein Symbol um:

1. Klicken Sie im Quelltext-Editor auf den Bezeichner, der umbenannt werden soll. Bei dem Bezeichner kann es sich um eine Methode, eine Variable, ein Feld, eine Klasse, einen Record, eine Struktur, ein Interface, einen Typ oder einen Parameter handeln.
2. Wählen Sie im Hauptmenü oder im Kontextmenü des Quelltext-Editors den Befehl **Refactor > Umbenennen**.
3. Geben Sie im Dialogfeld Umbenennen den neuen Bezeichner in das Feld Neuer Name ein.
4. Lassen Sie das Kontrollkästchen Vor Refactoring Referenzen anzeigen aktiviert. Wenn diese Option deaktiviert ist, wird das Refactoring ohne Vorschau der Änderungen sofort ausgeführt.
5. Klicken Sie auf OK. Das Dialogfeld Refactorings wird bei jedem Vorkommen des zu ändernden Bezeichners angezeigt.
6. Überprüfen Sie die vorgeschlagenen Änderungen im Dialogfeld Refactorings, und klicken Sie oben im Dialogfeld auf die Schaltfläche Refactor, um die aufgeführten Refactoring-Operationen durchzuführen. Mit Hilfe der Schaltfläche Refactoring entfernen können Sie die ausgewählte Refactoring-Operation aus dem Dialogfeld entfernen.

So deklarieren Sie eine Variable:

1. Klicken Sie im Quelltext-Editor auf ein noch nicht deklarierte Variable.

Anmerkung: Alle undeclareden Variablen werden von der Fehlermarkierungsfunktion mit einer roten Wellenlinie unterstrichen.

2. Wählen Sie im Hauptmenü oder im Kontextmenü des Quelltext-Editors den Befehl **Refactor > Variable deklarieren**. Wurde die Variable im selben Gültigkeitsbereich bereits deklariert, steht der Befehl nicht zur Verfügung.
3. Legen Sie die Einstellungen im Dialogfeld Neue Variable deklarieren nach Bedarf fest.
4. Klicken Sie auf OK.

Die Variablendeclaration wird der Prozedur hinzugefügt. Sie basiert auf den Werten, die Sie im Dialogfeld Neue Variable deklarieren eingegeben haben.

So deklarieren Sie ein Feld:

1. Klicken Sie im Quelltext-Editor auf ein noch nicht deklariertes Feld.
2. Wählen Sie im Hauptmenü oder im Kontextmenü des Quelltext-Editors den Befehl **Refactor > Feld deklarieren**.
3. Legen Sie die Einstellungen im Dialogfeld Neues Feld deklarieren nach Bedarf fest.
4. Klicken Sie auf OK.

Die neue Felddeclaration wird dem type-Abschnitt des Quelltextes hinzugefügt. Sie basiert auf den Werten, die Sie im Dialogfeld

Neues Feld deklarieren eingegeben haben.

Anmerkung: Wenn es zwischen dem neuen und einem anderen Feld in demselben Gültigkeitsbereich zu einem Konflikt kommt, wird das Dialogfeld Refactorings geöffnet. Darin werden Sie aufgefordert, das Problem vor dem Fortsetzen zu beheben.

So erstellen Sie eine Methode aus einem Codefragment:

1. Wählen Sie im Quelltext-Editor das Codefragment aus, das in eine Methode extrahiert werden soll.
2. Wählen Sie im Hauptmenü oder im Kontextmenü des Quelltext-Editors den Befehl **Refactor > Methode extrahieren**. Das Dialogfeld Methode extrahieren wird geöffnet.
3. Geben Sie in das Feld Neue Methode einen Namen für die Methode ein, oder übernehmen Sie den vorgeschlagenen Namen.
4. Überprüfen Sie den Quelltext im Fenster Extrahierter Beispielcode.
5. Klicken Sie auf OK.

RAD Studio verschiebt den extrahierten Code an eine Position außerhalb der aktuellen Methode, legt die erforderlichen Parameter fest, erzeugt nach Bedarf lokale Variablen, bestimmt den Rückgabetyp und ersetzt das ursprüngliche Codefragment durch einen Aufruf der neuen Methode.

So konvertieren Sie eine String-Konstante in einen Ressourcen-String (nur Delphi):

1. Wählen Sie im Quelltext-Editor den in Anführungszeichen eingeschlossenen String aus, der in einen Ressourcen-String konvertiert werden soll. Im folgenden Code müssten Sie den Cursor beispielsweise in die Konstante `Hello World` setzen:

```
procedure foo;
begin
  writeln('Hello World');
end;
```

2. Wählen Sie im Hauptmenü oder im Kontextmenü des Quelltext-Editors den Befehl **Refactor > Ressourcen-String extrahieren**.

Anmerkung: Sie können stattdessen auch die Tastenkombination UMSCHALT+STRG+L drücken.

Das Dialogfeld Ressourcen-String extrahieren wird geöffnet.

3. Geben Sie einen Namen für den Ressourcen-String ein, oder übernehmen Sie den vorgeschlagenen Namen (Str gefolgt von dem String).
4. Klicken Sie auf OK.

Das Schlüsselwort `resourcestring` und der Ressourcenstring werden zum **implementation**-Abschnitt des Codes hinzugefügt, und der Originalstring wird durch den neuen Ressourcen-String ersetzt.

```
resourcestring
  strHelloWorld = 'Hello World';

procedure foo;
begin
  writeln(StrHelloWorld);
end.
```

So suchen Sie nach Namespaces oder Units und fügen sie der `uses`-Klausel hinzu:

1. Klicken Sie im Quelltext-Editor auf die Variable, deren Unit Sie der `uses`-Klausel (Delphi) bzw. deren Namespace Sie der `using`-Klausel (C#) hinzufügen möchten.
2. Wählen Sie im Hauptmenü oder im Kontextmenü des Quelltext-Editors **Refactor > Unit suchen**. Das Dialogfeld Unit suchen mit einer Liste der verfügbaren Delphi-Units wird angezeigt.

Anmerkung: In C# hat dieses Dialogfeld den Namen Namespace verwenden.

3. Wählen Sie die Unit oder den Namespace aus, die/der zur `uses`- bzw. `using`-Klausel im aktuellen Gültigkeitsbereich hinzugefügt werden soll. Sie können beliebig viele Units bzw. Namespaces auswählen.
4. Wenn Sie mit Delphi arbeiten, wählen Sie, wo die Referenz eingefügt wird (entweder im Abschnitt **interface** oder im Abschnitt **implementation**).

Anmerkung: Diese Auswahl ist in C# nicht von Bedeutung und wird daher nicht angeboten.

5. Klicken Sie auf OK.

Die `uses`- oder `using`-Klausel wird mit den ausgewählten Units oder Namespaces aktualisiert.

Siehe auch

[Überblick zum Refactoring \(siehe Seite 1412\)](#)

10.4.5 Überblick zum Extrahieren von Ressourcen-Strings (Delphi)

Durch das Extrahieren von Ressourcen-Strings lassen sich String-Definitionen zusammenfassen, um ihre Übersetzung zu vereinfachen. Sie können String-Werte in Ressourcen-Strings extrahieren, die im `resourcestring`-Abschnitt einer Quelltextdatei definiert sind. Enthält die Datei keinen `resourcestring`-Abschnitt, wird er von der Refactoring-Engine automatisch erzeugt und nach dem Schlüsselwort **implementation** oder der **uses**-Liste eingefügt.

Aus folgenden Elementen können keine Ressourcen-Strings erzeugt werden:

- **Konstanten.** Beispielsweise lässt sich `const A = 'abcdefg';` nicht in einen Ressourcen-String extrahieren.
- **Konstanten in Parametern.** Der String `MyProc(A, B: Integer; C: string='test');` kann z.B. nicht in einen Ressourcen-String extrahiert werden.
- **Ressourcen-Strings.** Beispielsweise ist `resourcestring A = 'test';` bereits ein Ressourcen-String, der nicht extrahiert werden kann.

Siehe auch

[Überblick zum Refactoring \(siehe Seite 1412\)](#)

[Refactoring von Quelltext \(siehe Seite 1415\)](#)

10.4.6 Refactoring-Operationen in der Vorschau anzeigen und durchführen

Bei den meisten Refactoring-Operationen können die Auswirkungen vor der Durchführung im Vorschaufenster angezeigt werden. Einige Operationen unterstützen jedoch keine Vorschau und werden sofort durchgeführt. Wenn Sie zum ersten Mal ein Refactoring durchführen, sollten Sie unbedingt die Möglichkeit zur Vorschau nutzen. Im Vorschaufenster wird angezeigt, wie die Refactoring-Engine Operationen für die verschiedenen Arten von Symbolen und anderen Refactoring-Zielen auswertet und ausführt. Die Vorschau wird per Voreinstellung angezeigt (sofern möglich). Während der Vorschau einer Operation sammelt die Engine in einem Hintergrund-Thread Refactoring-Informationen und zeigt diese sofort an.

Die Verarbeitung in einem Hintergrund-Thread erfolgt auch bei Refactoring-Operationen, die sofort (ohne Vorschau) durchgeführt werden. Hier wird lediglich durch ein modales Dialogfeld die Anzeige unterdrückt. Falls während des Sammelns von Informationen ein Fehler auftritt, führt die Refactoring-Engine die Operation nicht durch. Refactoring-Operationen werden

von der Engine nur umgesetzt, wenn die Informationssammlung fehlerlos abgeschlossen werden konnte.

So zeigen Sie die Vorschau einer Refactoring-Operation an:

1. Öffnen Sie ein Projekt.
2. Suchen Sie im Quelltext-Editor nach einem Symbolnamen.
3. Wählen Sie den Symbolnamen aus.
4. Klicken Sie mit der rechten Maustaste, um das Kontextmenü zu öffnen.
5. Wählen Sie **Refactoring**►**Symbol umbenennen 'Symbolname'**, wobei **Symbolname** der Name des ausgewählten Symbols ist. Das Dialogfeld **Symbol umbenennen** wird geöffnet.
6. Geben Sie einen neuen Namen in das Feld **Neuer Name** ein.
7. Aktivieren Sie das Kontrollkästchen **Vor Refactoring Referenzen anzeigen**.
8. Klicken Sie auf **OK**. Eine hierarchische Liste der Elemente, die potenziell vom Refactoring betroffen sind, wird angezeigt. Die Elemente sind chronologisch in der Reihenfolge aufgeführt, in der sie gefunden wurden. Sie können im Quelltext-Editor zu den einzelnen Elementen wechseln.

Anmerkung: Soll ein Element in der Refactoring-Operation nicht berücksichtigt werden, wählen Sie es aus und klicken in der Symbolleiste auf das Symbol **Refactoring löschen**.

So gelangen Sie vom Meldungsfenster aus zu einem Refactoring-Ziel:

1. Erweitern Sie einen der Knoten im Meldungsfenster.
2. Klicken Sie auf das Refactoring-Ziel, das im Quelltext-Editor angezeigt werden soll.
3. Nehmen Sie die erforderlichen Änderungen im Quelltext-Editor vor.

Warnung: Wenn Sie ein Element im Quelltext-Editor ändern, wird die Refactoring-Operation blockiert. Wenn Sie Änderungen an Dateien vornehmen, während das Meldungsfenster Refactoring-Ziele enthält, müssen Sie die Refactoring-Operation anschließend erneut durchführen.

So führen Sie eine Refactoring-Operation durch:

1. Öffnen Sie ein Projekt.
2. Suchen Sie im Quelltext-Editor nach einem Symbolnamen.
3. Wählen Sie den Symbolnamen aus.
4. Klicken Sie mit der rechten Maustaste, um das Kontextmenü zu öffnen.
5. Wählen Sie **Refactoring**►**Symbol umbenennen 'Symbolname'**, wobei **Symbolname** der Name des ausgewählten Symbols ist. Das Dialogfeld **Symbol umbenennen** wird geöffnet.
6. Geben Sie einen neuen Namen in das Feld **Neuer Name** ein.
7. Klicken Sie auf **OK**. Wenn das Kontrollkästchen **Vor Refactoring Referenzen anzeigen** nicht aktiviert ist, wird das Refactoring sofort ausgeführt.

Warnung: Falls die Engine während der Operation Fehler feststellt, wird das Refactoring nicht durchgeführt. Die Fehler werden im Meldungsfenster angezeigt.

Siehe auch

Überblick zum Refactoring (siehe Seite 1412)

10.4.7 Überblick zum Deklarieren von Variablen und Feldern (Delphi)

Sie können mit Hilfe der Refactoring-Funktion Variablen und Felder erstellen. Dadurch können Variablen und Felder während der Programmierung ohne weitere Vorausplanung deklariert werden. Dieses Thema enthält Informationen zu folgenden Bereichen:

- Variablen deklarieren
- Vorschlag für den VariablenTyp
- Felder deklarieren

Variablen deklarieren

Sie können für einen nicht deklarierten Bezeichner innerhalb des Gültigkeitsbereichs eines Prozedurblocks eine Variable erstellen. Sie brauchen den Bezeichner dazu lediglich auszuwählen und einen Menübefehl aufzurufen bzw. eine Tastenkombination zu drücken. Das Dialogfeld Variable deklarieren enthält nach dem Öffnen einen Vorschlag für den Variablenamen, der auf der Auswahl basiert. Wenn Sie der Variablen einen anderen Namen zuweisen, wird sie erstellt, jedoch wird das Symbol für einen nicht deklarierten Bezeichner (Fehlermarkierung durch Unterstreichung) weiterhin angezeigt.

Variablennamen müssen den Sprachregeln für Bezeichner entsprechen. In Delphi gelten für Variablennamen folgende Regeln:

- Der Name darf kein Schlüsselwort sein.
- Der Name darf keine Leerzeichen enthalten.
- Der Name darf kein reserviertes Wort wie **if** oder **begin** sein.
- Der Name muss mit einem alphabetischen Unicode-Zeichen oder einem Unterstrich beginnen und kann alphanumerische Unicode-Zeichen oder weitere Unterstriche enthalten.
- In Delphi kann als Typname das Schlüsselwort **string** verwendet werden.

Anmerkung: Im .NET SDK wird empfohlen, Bezeichner nicht mit Unterstrichen zu beginnen, da dieses Namensmuster für Systemzwecke reserviert ist.

Anmerkung: Im Dialogfeld zur Variablen Deklaration können Sie einen Anfangswert für die Variable festlegen.

Vorschlag für den VariablenTyp

Die Refactoring-Engine schlägt einen Typ für die zu erstellende Variable vor. Dazu wertet sie binäre Operationen für die ausgewählte Anweisung aus und verwendet den Typ der Summe der untergeordneten Operanden als Typ für die neue Variable. Betrachten Sie z.B. die folgende Anweisung:

```
myVar := x + 1;
```

Die Refactoring-Engine setzt automatisch voraus, dass die neue Variable *myVar* den Typ Integer haben soll, wenn *x* ein Integer-Wert ist.

Oft kann die Refactoring-Engine den Typ durch die Auswertung der Anweisung ableiten. Beispielsweise impliziert die Anweisung **If foo Then...**, dass *foo* vom Typ Boolean ist. Bei der Anweisung **If (foo = 5) Then...** ergibt die Auswertung den Typ Boolean, obwohl es sich hier um den Vergleich einer Ordinalzahl (5) mit einem unbekannten Typ (*foo*) handelt. Die binäre Operation zeigt an, dass *foo* ein Ordinalwert sein muss.

Felder deklarieren

Sie können für einen nicht deklarierten Bezeichner innerhalb des Gültigkeitsbereichs einer Klasse ein Feld deklarieren. Ähnlich wie eine Variable kann auch ein Feld im Quelltext mittels Refactoring deklariert werden. Die Refactoring-Engine generiert die

Felddeklaration automatisch an der richtigen Stelle. Voraussetzung für den Erfolg der Operation ist, dass sich das Feld innerhalb des Gültigkeitsbereichs seiner übergeordneten Klasse befindet. Zu diesem Zweck können Sie das Feld innerhalb der Klasse erstellen oder seinem Namen den Namen des Objekts voranstellen, das den Kontext für das Feld bereitstellt.

Hinsichtlich des Namens gelten bei der Felddeklaration dieselben Regeln wie für Variablen:

- Der Name darf kein Schlüsselwort sein.
- Der Name darf keine Leerzeichen enthalten.
- Der Name darf kein reserviertes Wort wie **if** oder **begin** sein.
- Der Name muss mit einem alphabetischen Unicode-Zeichen oder einem Unterstrich beginnen und kann alphanumerische Unicode-Zeichen oder weitere Unterstriche enthalten.
- In Delphi kann als Typname das Schlüsselwort **string** verwendet werden.

Anmerkung: In .NET sind Bezeichner, die mit einem Unterstrich beginnen, für Systemzwecke reserviert.

Sie können eine Sichtbarkeit für das Feld festlegen. Wenn Sie eine andere Sichtbarkeit als **private** oder **strict private** wählen, führt die Refactoring-Engine folgende Operationen durch:

- Suche nach allen untergeordneten Klassen.
- Durchsuchen jeder untergeordneten Klasse zur Bestimmung des Feldnamens.
- Anzeigen eines roten Fehlersymbols, wenn der Feldname zu einem Konflikt mit einem Feld in einer abgeleiteten Klasse führt.
- Das Refactoring ist nicht möglich, wenn ein Konflikt mit einem vorhandenen Elementnamen auftritt.

Refactoring-Beispiel

Die folgenden Beispiele zeigen, was geschieht, wenn Variablen und Felder mit der Refactoring-Funktion deklariert werden.

Sehen Sie sich folgenden Quelltext an:

```
TFoo = class
private
  procedure Fool;
end;
...

implementation

procedure TFoo.Fool;
begin
  FTestString := 'test';    // Refactoring von TString, Feld zuweisen
end;
```

Angenommen, Sie deklarieren ein Feld mittels Refactoring. Sie erhalten dann folgendes Ergebnis:

```
TFoo = class
private
  FTestString: string;
  procedure Fool;
end;
```

Wenn Sie stattdessen eine Variable deklarieren, lautet das Ergebnis:

```
procedure TFoo.Fool;
var
  TestString: string;    // Hinzugefügt durch Refactoring
begin
  TestString := 'test';    // Hinzugefügt durch Refactoring
  TestString := 'whatever';
end;
```

Siehe auch

[Überblick zum Refactoring \(siehe Seite 1412\)](#)

[Überblick zum Umbenennen von Symbolen \(siehe Seite 1413\)](#)

[Refactoring von Quelltext \(siehe Seite 1415\)](#)

10.4.8 Überblick zum Suchen von Referenzen (Delphi, C#, C++)

Manchmal möchten Sie den Quelltext nicht ändern, sondern Referenzen auf einen bestimmten Bezeichner suchen. In der Refactoring-Engine stehen die Befehle Referenzen suchen, Lokale Referenzen suchen und Deklarationssymbol suchen zur Verfügung.

Bei Auswahl von Referenzen suchen oder Lokale Referenzen suchen wird das Fenster Referenzen suchen-Fenster geöffnet, das eine hierarchische Liste mit allen Instanzen der gewählten Referenz enthält. Der Befehl Referenzen suchen liefert ein Baumdiagramm mit allen Referenzen im gesamten Projekt. Wenn Sie nur die lokalen Referenzen (in der aktiven Quelltextdatei) anzeigen möchten, wählen Sie im Menü Suchen den Befehl Lokale Referenzen suchen. Um die ursprüngliche Deklaration in der aktiven Quelltextdatei anzuzeigen, wählen Sie Deklarationssymbol suchen. Den Befehl Deklarationssymbol suchen gibt es nur in Delphi und nicht in C#.

Refactoring-Beispiel

Das folgende Beispiel zeigt, wie das Suchen nach Referenzen durchgeführt wird:

```
1 TFOO = class
2   loc_a: Integer;           // Die Suche nach Referenzen auf loc_a findet
3   procedure Foo;            // nur diese Zeile (Zeile 2) und die Verwendung
4 end;                      // in TFOO.Foo (Zeile 15)

5 var
6   loc_a: string;           // Diese Suche nach Referenzen auf loc_a
7   implementation           // findet nur diese Zeile (Zeile 6) und
8   {$R *.nfm}                // die Verwendung in Prozedur Foo (Zeile 11)

9 procedure Foo;
10 begin
11   loc_a := 'test';
12 end;

13 procedure TFOO.Foo;
14 begin  ///
15   loc_a:=1;
16 end;
```

Siehe auch

[Überblick zum Refactoring \(siehe Seite 1412\)](#)

[Refactoring von Quelltext \(siehe Seite 1415\)](#)

10.4.9 Überblick zum Ändern von Parametern (Delphi)

Das Hinzufügen oder Entfernen von Funktionsparametern gehört zu den häufig ausgeführten und aufwändigen Programmieraufgaben. In RAD Studio kann dieser Vorgang mit Hilfe der Refactoring-Funktion Parameter ändern automatisiert werden. Sie können Funktionsparameter hinzufügen, entfernen und neu anordnen.

Um das Refactoring auszuführen, markieren Sie im Quelltext-Editor einen Funktionsnamen und wählen **Refactor ▶ Params ändern**.

Bei Anwendung der Refactoring-Funktion Parameter ändern können in folgenden Situationen Konflikte bezüglich der Funktionssignatur auftreten:

- Die Funktion, für die Sie das Refactoring durchführen, wird in einer abgeleiteten Klasse überschrieben (**override**). Das Refactoring betrifft in diesem Fall auch alle überschreibenden Funktionen.
- Eine abgeleitete Klasse enthält eine überladene Version der Funktion mit derselben Signatur wie die Version, für die das Refactoring durchgeführt wird. Beim Refactoring der Funktion wird die Direktive **overload** durch **override** ersetzt.
- Eine abgeleitete Klasse enthält eine überschriebene Methode, die mit der Originalsignatur identisch ist. Beim Refactoring der Funktion wird die Direktive **override** durch **overload** ersetzt.

Anmerkung: Nachdem ein Parameter entfernt wurde, muss der Quelltext von Methoden, in denen dieser Parameter verwendet wird, manuell gelöscht werden.

Siehe auch

Überblick zum Refactoring (siehe Seite 1412)

10.4.10 Sync-Bearbeitungsmodus (Delphi, C#, C++)

Im Sync-Bearbeitungsmodus werden Änderungen, die Sie an einem Bezeichner vornehmen, für alle anderen Instanzen dieses Bezeichners wirksam. Nach der Aktivierung des Sync-Bearbeitungsmodus können Sie zu jedem hervorgehobenen Bezeichner im aktuellen Fenster des Quelltext-Editors wechseln. Wenn Sie einen Bezeichner ändern, der mehrfach in der Datei vorhanden ist, werden alle Instanzen des Bezeichners synchron geändert.

Siehe auch

Überblick zum Refactoring (siehe Seite 1412)

Sync-Bearbeitungsmodus verwenden (siehe Seite 54)

Refactoring von Quelltext (siehe Seite 1415)

10.4.11 Refactoring rückgängig machen (Delphi, C#)

Die Refactoring-Engine nutzt einen Versionierungsmechanismus namens *Local Striping*, um das Rückgängigmachen von Namensänderungen in Quelltextdateien zu ermöglichen. Die IDE zeichnet für jede im aktuellen Refactoring-Änderungssatz den aktuellen Zeitstempel auf. Der Zeitstempel kennzeichnet jeweils eine bestimmte lokale Version der Datei. Wenn Sie den Befehl Rückgängig auswählen, überschreibt die IDE die Datei, für die das Refactoring durchgeführt wurde, durch die lokale Sicherungsdatei mit dem entsprechenden Zeitstempel.

Zusätzliche Änderungen, die nach dem Refactoring an Dateien vorgenommen wurden, werden durch Auswahl von Rückgängig ebenfalls rückgängig gemacht. Deshalb wird vor jeder Rückgängig-Aktion eine Warnmeldung angezeigt, in der Sie zur Bestätigung der Aktion aufgefordert werden. Wenn Sie die Rückgängig-Aktion durchführen, wird für alle geänderten Dateien der Zustand wiederhergestellt, den sie vor dem Refactoring hatten. Dabei gehen auch alle Änderungen verloren, die nach dem Refactoring an diesen Dateien vorgenommen wurden.

Es werden nur Änderungen zurückgenommen, die mit der Umbenennungsoperation der Refactoring-Funktion vorgenommen wurden, da sich nur diese Refactoring-Operation auf mehrere Dateien auswirkt.

Die Auswirkungen von Refactoring-Operationen wie Methode extrahieren, Feld deklarieren oder Variable deklarieren können mit STRG+Z (dem regulären Rückgängig-Mechanismus) im Quelltext-Editor oder mit der Schaltfläche Rückgängig im Fenster Refactoring rückgängig gemacht werden.

Siehe auch

Überblick zum Refactoring (siehe Seite 1412)

Refactoring von Quelltext (siehe Seite 1415)

10.4.12 Referenzen suchen

Mit Hilfe der Refactoring-Funktion zum Suchen von Referenzen können Sie Verbindungen suchen, die zwischen einer Datei, die ein Symbol für eine geplante Umbenennung enthält, und anderen Dateien bestehen, die ebenfalls dieses Symbol enthalten. In einer Vorschau lässt sich bestimmen, wie beim Refactoring mit bestimmten Zielen oder einer Gruppe von Referenzen verfahren werden soll.

So erstellen Sie eine Suchliste:

1. Öffnen Sie ein Projekt.
2. Wählen Sie einen Bezeichner im Quelltext-Editor aus.
3. Wählen Sie **Suchen > Referenzen suchen**.

Anmerkung: Sie können diese Funktion auch mit dem Tastaturkürzel UMSCHALT+STRG+EINGABE aufrufen.

4. Doppelklicken Sie im Fenster auf einen Knoten, um an die entsprechende Position im Quelltext-Editor zu gelangen.

Anmerkung: Wenn Sie die Suche nach Referenzen fortsetzen, ohne Ergebnisse zu löschen, werden alle weiteren Ergebnisse in chronologischer Reihenfolge an die vorhandenen Ergebnisse im Fenster angefügt.

So löschen Sie Ergebnisse im Fenster zur Referenzsuche:

1. Wählen Sie eine einzelne Referenz oder einen Knoten aus.

Anmerkung: Sie erhalten nun unabhängig von Ihrer Auswahl dasselbe Ergebnis. Der gesamte Knoten wird gelöscht.

2. Klicken Sie auf das Symbol Löschen  oben im Fenster Referenzen suchen, um das ausgewählte Element sowie alle Elemente, die in der Hierarchie darunter liegen, zu löschen.

Anmerkung: Elemente, die im Fenster Referenzen suchen gelöscht werden, bleiben aber in den Quelltextdateien oder im Projekt erhalten.

So löschen Sie alle Ergebnisse im Fenster zur Referenzsuche:

1. Wählen Sie ein Element im Fenster aus.
2. Klicken Sie oben im Fenster Referenzen suchen auf das Symbol Alle Referenzen entfernen . Alle Ergebnisse im Fenster

werden nun gelöscht.

Anmerkung: Elemente, die im Fenster Referenzen suchen gelöscht werden, bleiben aber in den Quelltextdateien oder im Projekt erhalten.

Siehe auch

Überblick zum Refactoring (siehe Seite 1412)

Überblick zum Suchen von Referenzen (siehe Seite 1421)

10.4.13 Units suchen und Namespaces verwenden (Delphi, C#)

Je nach der verwendeten Programmiersprache können Sie mit einer Refactoring-Funktion nach Namespaces oder Units suchen. Wenn Sie C# verwenden, können Sie mit dem Befehl Namespace verwenden anhand eines Objekts im Quelltext Namespaces in Ihre Quelltextdateien importieren. In Delphi können Sie mit dem Befehl Unit suchen anhand der Objekte im Quelltext nach Units suchen und diese zu Ihren Quelltextdateien hinzufügen.

Das Dialogfeld Namespace verwenden wird geöffnet, wenn Sie einen C#-Objektnamen auswählen und auf den Befehl **Namespace verwenden** klicken. Die Importfunktion versucht, die wahrscheinlichsten Namespaces zu ermitteln und anzuzeigen. Sie können mehrere Namespaces für das Hinzufügen zur **using**-Klausel auswählen. Die Funktion ist in Delphi identisch. Sie versucht, die entsprechende Unit mit der Definition des ausgewählten Objekts zu finden und fügt die markierte Unit zur **uses**-Klausel hinzu.

Siehe auch

Überblick zum Refactoring (siehe Seite 1412)

Refactoring von Quelltext (siehe Seite 1415)

10.5 Anwendungen testen

Das Testen von Units ist ein wesentlicher Schritt bei der Entwicklung stabiler Anwendungen. In den folgenden Themen werden die Funktionen zum Testen von Units in RAD Studio vorgestellt.

10.5.1 Überblick über das Testen von Units

In RAD Studio sind die zwei Open Source Test-Frameworks *DUnit* und *NUnit* integriert, mit denen Sie automatisierte Testfälle für Ihre Anwendungen erstellen und ausführen können. Das DUnit-Framework steht für Delphi und C++ zur Verfügung. Das NUnit-Framework steht nur für Delphi für .NET und C# zur Verfügung. Diese Frameworks vereinfachen die Entwicklung von Tests für die Klassen und Methoden einer Anwendung. Die Unit-Tests können zusammen mit den Refactoring-Funktionen die Stabilität Ihrer Anwendungen verbessern. Wenn Sie nach jeder Quelltextänderung bestimmte Standardtests ausführen, ist die Wahrscheinlichkeit größer, dass Sie Probleme früh im Entwicklungszyklus erkennen.

Beide Test-Frameworks basieren auf JUnit und verfügen zum größten Teil über identische Funktionen.

Dieses Thema enthält folgende Informationen:

- Was installiert wird
- Testprojekte
- Testfälle
- Testgruppen

Was installiert wird

Standardmäßig werden bei der vollständigen Installation von RAD Studio beide Frameworks installiert. Bei der Installation von einzelnen Personalities werden die von diesen Personalities unterstützten Frameworks installiert.

DUnit

Für Delphi und C++Builder wird das DUnit-Framework automatisch vom RAD Studio-Installationsprogramm installiert. Viele DUnit-Ressourcen befinden sich im Verzeichnis `\source\DUnit` unter dem Stamminstallationsverzeichnis. Dort finden Sie beispielsweise die Quelltextdateien, die Dokumentation und die Testbeispiele. Für C++Builder werden auch die folgenden C++-Header- und Bibliotheksdateien für die Verwendung als C++-Testprojekte bereitgestellt:

- GUITestRunner.hpp
- XMLTestRunner.hpp
- TextTestRunner.hpp
- TestFramework.hpp
- DUnitMainForm.hpp
- DUnitAbout.hppdir
- dunitrtl.lib

Anmerkung: Diese Dateien sind kein Bestandteil der DUnit-Standarddistribution. Sie wurden von CodeGear vorerzeugt und für Sie in C++Builder bereitgestellt.

Wenn Sie DUnit verwenden, sollten Sie im Allgemeinen mindestens einen Testfall und eine oder mehrere Testgruppen

einbeziehen. Die Testfälle enthalten in der Regel eine oder mehrere Annahmeanweisungen, um die Funktionsweise der getesteten Klasse zu überprüfen.

DUnit unterliegt der Mozilla Public License 1.0 (MPL).

NUnit

Das NUnit -Framework steht nur für die Personalities Delphi für .NET und C# zur Verfügung.

Während der Installation werden Sie gefragt, ob NUnit installiert werden soll. Sie können es annehmen oder ablehnen. Sie können das vorgeschlagene Verzeichnis für NUnit ändern oder übernehmen. Im letzten Fall wird NUnit im Verzeichnis `C:\Programme\NUnit v2.x` installiert (x ist die Versionsnummer). Das Installationsverzeichnis enthält eine Vielzahl von Ressourcen, einschließlich Dokumentation und Beispieltests.

NUnit ist das .NET-Test-Framework und kann zum Testen von Delphi für .NET- und von C#-Projekten verwendet werden. Es gibt einige feine, aber wichtige Unterschiede zwischen der Funktionsweise von NUnit und DUnit. So erstellt beispielsweise NUnit beim Linken keine .dcu-Dateien wie DUnit.

Wenn Sie NUnit verwenden, sollten Sie im Allgemeinen mindestens einen Testfall und eine oder mehrere Testgruppen einbeziehen. Die Testfälle enthalten in der Regel eine oder mehrere Annahmeanweisungen, um die Funktionsweise der getesteten Klasse zu überprüfen.

Testprojekte

Ein Testprojekt kapselt einen oder mehrere Testfälle und wird in der Projektverwaltung durch einen Knoten angezeigt. RAD Studio stellt den Testprojekt-Experten zur Verfügung, mit dem Sie ein Basistestprojekt erstellen können. Nachdem ein Testprojekt einem Quelltextprojekt zugeordnet ist, können Sie Testfälle erstellen und sie dem Testprojekt hinzufügen.

Testfälle

In einem typischen Unit-Testprojekt hat jede zu testende Klasse eine zugehörige Testklasse; das ist aber nicht erforderlich. Die Testklasse wird auch *Testfall* genannt. Abhängig von dem verwendeten Framework kann die Testklasse von einer speziellen Testfallbasisklasse abgeleitet sein. Im Allgemeinen verfügt ein Testfall über eine oder mehrere Methoden, die den Methoden in der zu testenden Klasse entsprechen. In einem Testprojekt können mehrere Testfälle aufgenommen werden. Diese Möglichkeit zum Gruppieren und Kombinieren von Tests in Testfällen — und Testfälle in Testprojekten — macht den Unterschied zwischen einem Testfall und den einfachen Formen des Testens (z.B. Verwenden von Ausgabeanweisungen oder Auswerten von Debugger-Ausdrücken) aus. Jeder Testfall und jedes Testprojekt kann wiederverwendet, erneut ausgeführt und durch die Verwendung von Stapeldateien, Build-Scripts oder anderen Typen von Testsystemen automatisiert werden.

Sie sollten Ihre Tests normalerweise in einem eigenen Projekt getrennt vom Quelltextprojekt erstellen. Sie brauchen die Tests dann nicht aus der fertigen Anwendung zu entfernen. RAD Studio stellt den Testfall-Experten bereit, der Sie beim Erstellen der Basistestfälle unterstützt, die Sie dann nach Bedarf anpassen können.

Testgruppen

Der Begriff *Testgruppe* bezieht sich auf die Kombination mehrerer Testfälle, die logisch zusammenhängende Funktionsweisen testen. Sie können Testgruppen in Ihrem Testfall definieren. In der Regel führen Sie die Instantiierung der Objekte, Initialisierung der Variablen, Einrichtung der Datenbankverbindung und andere Wartungsaufgaben in den Abschnitten `SetUp` und `TearDown` durch. Solange die Tests mit denselben Objekten durchgeführt werden, können Sie beliebig viele Tests zu einer Testgruppe hinzufügen.

Siehe auch

Überblick über DUnit (siehe Seite 1429)

Überblick über NUnit (siehe Seite 1432)

Entwickeln von Tests (siehe Seite 1427)

[Mozilla Public Lizenz 1.0](#)

[zlib/libpng Lizenz](#)

10.5.2 Entwickeln von Tests

Die Struktur eines Unit-Tests ist größtenteils von der Funktionsweise der getesteten Klasse und Methode abhängig. Die Experten zum Testen von Units erzeugen für das Testprojekt Skeleton-Templates, setup- und teardown-Methoden und Basistestfälle. Anschließend können die Templates verändert werden, indem Sie eine spezielle Testlogik zum Testen von bestimmten Methoden hinzufügen.

Im Folgenden wird das Vorgehen beim Erstellen eines Unit-Testprojekts und eines Unit-Testfalls beschrieben. Führen Sie die Schritte in der angegebenen Reihenfolge durch. Das Unit-Testprojekt muss vor den zugehörigen Testfällen erstellt werden. Der Testfall-Experte steht nur zur Verfügung, wenn das aktuell aktive Projekt ein Unit-Testprojekt ist.

So erstellen Sie ein Testprojekt:

1. Wählen Sie **Datei > Neu > Weitere**.
2. Öffnen Sie den Ordner Unit-Test.
3. Doppelklicken Sie auf Testprojekt. Der Testprojekt-Experte wird geöffnet, und die Seite Details für das Testprojekt festlegen angezeigt.
4. Nehmen Sie die gewünschten Einstellungen vor, oder übernehmen Sie die Standardwerte. Geben Sie Folgendes ein:
 - Projektname: Geben Sie den Namen für das neue Testprojekt ein, oder übernehmen Sie die Vorgabe. Die Vorgabe ist der Name des aktiven Projekts mit dem an den Namen angehängten Wort *Tests*. Wenn kein aktives Projekt vorhanden ist, ist die Vorgabe *UnitTest* mit einer angehängten Sequenznummer.
 - Speicherort: Geben Sie den vollständigen Pfadnamen für den Ordner ein, in dem das Testprojekt erstellt werden soll, oder übernehmen Sie die Vorgabe. Die Vorgabe ist der Unterordner *test*, der sich unter dem aktiven Projektordner befindet. Wenn kein aktives Projekt vorhanden ist, ist die Vorgabe der Standardprojektordner. Klicken Sie auf die Ellipsen-Schaltfläche (...), um ein Durchsuchen Dialogfeld anzuzeigen, in dem Sie die Position auswählen können.
 - Personality: Wählen Sie aus der Dropdown-Liste die Personality (Codesprache) aus, oder übernehmen Sie die Vorgabe. Die Vorgabe ist die Personality des aktiven Projekts.
5. Wenn das Testprojekt nicht zu Ihrer Projektgruppe hinzugefügt werden soll, deaktivieren Sie das Kontrollkästchen Zur Projektgruppe hinzufügen.
6. Klicken Sie zum Fortsetzen auf Weiter oder auf Fertig stellen, um die restlichen Vorgaben zu übernehmen. Wenn die Schaltfläche Fertig stellen verfügbar ist, können Sie jederzeit darauf klicken, um die Vorgabewerte für die restlichen Felder zu übernehmen und das neue Testprojekt sofort zu erzeugen. Ansonsten klicken Sie auf Weiter, um zur Seite Optionen für das Test-Framework festlegen zu wechseln.
7. Nehmen Sie die gewünschten Einstellungen vor, oder übernehmen Sie die Standardwerte. Geben Sie Folgendes ein:
 - Test-Framework: Für die Delphi.Net-Personality können Sie entweder DUnit oder NUnit aus der Dropdown-Liste auswählen. Für die Delphi- und die C++-Personalities wird nur das DUnit-Framework unterstützt. Daher können Sie diesen Wert nicht ändern.. Für C# wird nur das NUnit-Framework unterstützt.
 - Test-Runner: Wählen Sie entweder GUI oder Konsole aus der Dropdown-Liste aus. Der Konsolen-Test-Runner leitet die Ausgabe an die Konsole. Der GUI Test-Runner zeigt die Ergebnisse interaktiv in einem GUI-Fenster mit Farbcodierungen zum Kennzeichnen von Erfolg oder Fehlschlag an.
8. Klicken Sie auf Fertig stellen. Der Testprojekt-Experte erzeugt die Testprojekt-Template. Für die Personalities Delphi und C# werden auch die erforderlichen Quelltextreferenzen in die Testprojekt-Template eingefügt; Sie können daher den nächsten Schritt überspringen. Für C++Builder müssen Sie das Testprojekt manuell mit den C++-Klassen, die getestet werden sollen, verknüpfen; fahren Sie in diesem Fall mit dem nächsten Schritt fort..
9. **Nur für C++Builder:** Verknüpfen Sie das neue Testprojekt mit dem zu testenden Quelltext. Bei C++Builder müssen Sie das Testprojekt manuell mit dem C++-Quelltext, der getestet werden soll, verknüpfen. Sie können dazu eine der folgenden Vorgehensweisen verwenden:

- Fügen Sie den C++-Quelltext direkt in das Testprojekt ein.
- Fügen Sie dem Testprojekt etwaige .obj-Dateien hinzu.
- Fügen Sie dem Testprojekt eine .lib-Datei (entweder eine statische Bibliothek oder eine Importbibliothek für eine DLL) hinzu .
- Beziehen Sie die Header-Datei ein, die die zu testende Klasse implementiert. Weitere Anleitungen finden Sie im nächsten Abschnitt **Dateien zu einem Projekt hinzufügen**.

Dateien zu einem Projekt hinzufügen

1. Öffnen Sie das Testprojekt, und aktivieren Sie es. Zum Aktivieren der Datei, wählen Sie sie in der Projektverwaltung aus und klicken auf die Schaltfläche Aktivieren.
2. Klicken Sie in der Projektverwaltung mit der rechten Maustaste auf den Namen des Testprojekts. Ein Kontextmenü mit Projektoptionen wird angezeigt.
3. Wählen Sie Hinzufügen.... Das Dateiauswahl-Dialogfeld Dem Projekt hinzufügen wird eingeblendet. Wählen Sie hier die Datei aus, die in Ihr Testprojekt einbezogen werden soll.
4. Wählen Sie die Datei aus, und klicken Sie auf OK. Damit wird die ausgewählte Datei dem Testprojekt hinzugefügt.

So erstellen Sie einen Testfall:

1. Klicken Sie auf das Register der Datei mit den Klassen, die Sie testen möchten. Die Datei wird dadurch im Quelltext-Editor aktiviert.
2. Wählen Sie **Datei > Neu > Weitere**.
3. Öffnen Sie den Ordner Unit-Test.
4. Doppelklicken Sie auf Testfall. Der Testfall-Experte wird geöffnet, und die Seite Die zu testenden Methoden auswählen angezeigt.
5. Geben Sie den Namen der Quelltextdatei ein, die die zu testenden Klassen enthält. Klicken Sie auf die Ellipsen-Schaltfläche (...), um ein Öffnen Dialogfeld anzuzeigen, in dem Sie die Datei auswählen können.
6. Wählen Sie die Klassen und Methoden aus, für die Sie Tests erstellen möchten. Klicken Sie in der Liste Verfügbare Klassen und Methoden auf das Kontrollfeld neben einem Element, um aus- bzw. abzuwählen. Standardmäßig sind alle Klassen und Methoden ausgewählt. Sie können einzelne Methoden in der Liste deaktivieren. Der Experte erzeugt lediglich für die ausgewählten Methoden Testfall-Templates. Wenn Sie eine Klasse ausschließen, ignoriert der Experte die gesamte Klasse und alle ihre Methoden, auch wenn Sie die Methoden nicht einzeln aus der Auswahl entfernt haben. Wenn Sie eine Klasse auswählen, aber keine ihrer Methoden markieren, erzeugt der Experte einen Testfall für die Klasse, generiert aber keine Testmethoden dafür.
7. Klicken Sie zum Fortsetzen auf Weiter oder auf Fertig stellen, um die restlichen Vorgaben zu übernehmen. Wenn die Schaltfläche Fertig stellen verfügbar ist, können Sie jederzeit darauf klicken, um die Vorgabewerte für die restlichen Felder zu übernehmen und den neuen Testfall sofort zu erzeugen. Ansonsten klicken Sie auf Weiter, um zur Seite Details für den Testfall festlegen zu wechseln.
8. Nehmen Sie die gewünschten Einstellungen vor, oder übernehmen Sie die Standardwerte.
 - Testprojekt: Wählen Sie das Testprojekt aus der Dropdown-Liste aus. Die Vorgabe ist das aktive Testprojekt; wenn Sie gerade mit dem Testprojekt-Experten ein Testprojekt erstellt haben, dann ist das neue Testprojekt die Vorgabe.
 - Dateiname : Geben Sie einen Dateinamen für den Testfall, den Sie erstellen ein, oder übernehmen Sie die Vorgabe. Die Vorgabe ist der Name der zu testenden Quelltextdatei mit dem Namenspräfix *Test*.
 - Test-Framework: Für die Delphi für .Net-Personality können Sie entweder DUnit oder NUnit aus der Dropdown-Liste auswählen. Für die Delphi für Win32- und die C++-Personalities wird nur das DUnit-Framework unterstützt. Daher können Sie diesen Wert nicht ändern.. Für C# wird nur das NUnit-Framework unterstützt.
 - Basisklasse:Wählen Sie aus der Dropdown-Liste die Basisklasse aus, oder übernehmen Sie die Vorgabe. Die Vorgabe ist *TTestCase*. Das ist die Standardbasisklasse von **TestCase**. In den meisten Fällen können Sie die Vorgabe verwenden. Sie können aber eine eigene benutzerdefinierte *TTestCase*-Klasse angeben. Außerdem können Sie beim Testen einer Objekthierarchie den neuen Testfall von der Klasse **TestCase** eines Basisobjekts der zu testenden Objekte ableiten. Dadurch kann eine abgeleitete Klasse einen Test erben, der für den Basistyp dieser Klasse erstellt wurde.

9. Klicken Sie auf Fertig stellen. Der Experte erzeugt eine Testfalldatei mit dem angegebenen Namen.

So schreiben Sie einen Testfall:

1. Fügen Sie den Methoden `SetUp` und `TearDown` in der Testfall-Template bei Bedarf Quelltext hinzu.
2. Fügen Sie den Testmethoden Annahmen hinzu.

So führen Sie einen Testfall im GUI-Test-Runner aus:

1. Aktivieren Sie die Datei, die die auszuführenden Klassen enthält. Wählen Sie die Datei in der Projektverwaltung aus, und klicken Sie auf die Schaltfläche Aktivieren.
2. Wählen Sie **Start > Start**. Der GUI-Test-Runner startet sofort bei der Ausführung Ihrer Anwendung.
3. Wählen Sie aus der Testliste einen oder mehrere Tests aus.
4. Klicken Sie auf die Schaltfläche Ausführen. Das Resultat der Tests wird im Fenster Testergebnisse angezeigt. Alle grün unterlegten Tests wurden erfolgreich abgeschlossen. Die rot markierten Tests sind fehlgeschlagen. Die gelb markierten Tests wurden nicht ausgeführt.
5. Überprüfen Sie die Testergebnisse.
6. Korrigieren Sie die Fehler im Quelltext, und führen Sie die Tests erneut aus.

Siehe auch

Überblick über das Testen von Units ( siehe Seite 1425)

Überblick über DUnit ( siehe Seite 1429)

Überblick über NUnit ( siehe Seite 1432)

10.5.3 Überblick über DUnit

DUnit ist ein Open Source Test-Framework, das auf JUnit basiert. Mit DUnit können Tests für Delphi Win32-Anwendungen erstellt und ausgeführt werden. Die RAD Studio-Integration des DUnit-Frameworks ermöglicht das Entwickeln und Ausführen von Tests für Delphi Win32-, Delphi .NET- und C++Builder-Anwendungen.

Jedes Test-Framework verfügt über eigene Methoden für das Testen von Bedingungen. Die Methoden entsprechen gebräuchlichen Annahmen. Sie können auch eigene Annahmen erstellen. Mit den bereitgestellten Methoden kann eine große Anzahl von Bedingungen getestet werden.

Dieses Thema enthält Informationen zu folgenden Bereichen:

- DUnit-Tests entwickeln
- DUnit-Funktionen
- DUnit-Test-Runner

DUnit-Tests entwickeln

Jeder DUnit-Test implementiert eine Klasse mit dem Typ `TTestCase`. Das folgende Delphi Win32-Beispielprogramm definiert zwei Funktionen, die eine einfache Addition und Subtraktion durchführen:

```
unit CalcUnit;

interface

type
  { TCalc }
```

```
TCalc = class
public
  function Add(x, y: Integer): Integer;
  function Sub(x, y: Integer): Integer;
end;

implementation

{ TCalc }

function TCalc.Add(x, y: Integer): Integer;
begin
  Result := x + y;
end;

function TCalc.Sub(X, Y: Integer): Integer;
begin
  Result := x + y;
end;

end.
```

Das folgende Beispiel zeigt die Testfall-Skeleton-Datei, die Sie ändern müssen, um die zwei Funktionen `Add` und `Sub` zu testen.

```
unit TestCalcUnit;

interface

uses
  TestFramework, CalcUnit;
type
  // Testmethoden für Klasse TCalc
  TestTCalc = class(TTestCase)
  strict private
    aTCalc: TCalc;
  public
    procedure SetUp; override;
    procedure TearDown; override;
  published
    procedure TestAdd;
    procedure TestSub;
  end;

implementation

procedure TestTCalc.SetUp;
begin
  aTCalc := TCalc.Create;
end;

procedure TestTCalc.TearDown;
begin
  aTCalc := nil;
end;

procedure TestTCalc.TestAdd;
var
  _result: System.Integer;
  y: System.Integer;
  x: System.Integer;
begin
  _result := aTCalc.Add(x, y);
  // TODO: Hier Testcode hinzufügen
end;

procedure TestTCalc.TestSub;
var
```

```

_result: System.Integer;
y: System.Integer;
x: System.Integer;
begin
  _result := aTCalc.Sub(x, y);
  // TODO: Hier Testcode hinzufügen
end;

initialization
  // Alle Testfälle bei Testprogramm registrieren
  RegisterTest(TestTCalc.Suite);
end.

```

DUnit-Funktionen

DUnit verfügt über verschiedene Funktionen, die Sie in Ihren Tests verwenden können.

Funktion	Beschreibung
Check	Prüft, ob eine Bedingung erfüllt wird.
CheckEquals	Prüft, ob zwei Elemente gleich sind.
CheckNotEquals	Prüft, ob zwei Elemente ungleich sind.
CheckNotNull	Prüft, ob ein Element ungleich Null ist.
CheckNull	Prüft, ob ein Element Null ist.
CheckSame	Prüft, ob zwei Elemente den gleichen Wert haben.
EqualsErrorMessage	Prüft, ob eine von der Anwendung ausgegebene Fehlermeldung mit einer bestimmten Fehlermeldung übereinstimmt.
Fail	Prüft, ob eine Routine fehlschlägt.
FailEquals	Prüft, ob ein Fehler einer bestimmten Fehlerbedingung entspricht.
FailNotEquals	Prüft, ob ein Fehler einer bestimmten Fehlerbedingung nicht entspricht.
FailNotSame	Prüft, ob zwei Fehlerbedingungen identisch sind.
NotEqualsErrorMessage	Prüft, ob zwei Fehlermeldungen identisch sind.
NotSameErrorMessage	Prüft, ob eine Fehlermeldung nicht mit einer bestimmten Fehlermeldung identisch ist.

Weitere Informationen zur Syntax und Verwendung dieser und anderer DUnit-Funktionen finden Sie in den DUnit-Hilfdateien im Verzeichnis [\source\dunit\doc](#).

DUnit-Test-Runner

Mit einem Testprogramm (Runner) können Sie die Tests unabhängig von Ihrer Anwendung ausführen. In einem DUnit-Testprojekt wird der Test-Runner-Code aus dem DUnit-Framework direkt in die erzeigte ausführbare Datei compiliert, so dass das Testprojekt zu einem Test-Runner wird. Dies unterscheidet sich vom NUnit-Framework, das einen separaten Test-Runner zur Ausführung von Tests verwendet. Das integrierte DUnit-Framework stellt zwei Test-Runner' bereit:

- Der *GUI Test-Runner* zeigt die Testergebnisse interaktiv in einem GUI-Fenster mit Farbcodierungen zum Kennzeichnen von Erfolg oder Fehlschlag an.
- Der *Konsolen-Test-Runner* leitet die Testausgabe an die Konsole.

Der DUnit GUI Test-Runner ist besonders hilfreich, wenn Sie interaktiv Unit-Tests oder den zu testenden Quelltext entwickeln. Die erfolgreichen Tests werden durch einen grünen, die fehlgeschlagenen Tests durch einen roten und die ausgelassenen Tests durch einen gelben Balken hervorgehoben.

Verwenden Sie den DUnit Konsolen-Test-Runner, wenn Sie fertig gestellten Quelltext testen oder die Tests über automatisierte Build-Scripts ausführen.

Siehe auch

Überblick über das Testen von Units (siehe Seite 1425)

Überblick über NUnit (siehe Seite 1432)

Entwickeln von Tests (siehe Seite 1427)

10.5.4 Überblick über NUnit

NUnit ist ein Open Source Test-Framework, das auf JUnit basiert. Mit NUnit können Tests für .NET Framework-Anwendungen entwickelt und ausgeführt werden. Durch die RAD Studio-Integration von NUnit können Sie Tests für Delphi für .NET-Anwendungen und C#-Anwendungen entwickeln und ausführen. Das NUnit-Framework wird nicht in C++Builder und Delphi für Win32 unterstützt.

Dieses Thema enthält Informationen zu folgenden Bereichen:

- Entwickeln von NUnit-Tests
- NUnit-Annahmen
- NUnit-Test-Runner

Entwickeln von NUnit-Tests

Jedes Test-Framework verfügt über eigene Methoden für das Testen von Bedingungen. Die Methoden entsprechen gebräuchlichen Annahmen. Sie können auch eigene Annahmen erstellen. Mit den bereitgestellten Methoden kann eine große Anzahl von Bedingungen getestet werden.

Wenn Sie eine Anwendung testen möchten, müssen Sie zuerst ein *Testprojekt* erstellen. Das Testprojekt enthält die *Testfalldateien* mit einem oder mehreren Tests. Ein Testfall entspricht einer Klasse. Jeder Test entspricht einer Methode. In der Regel erstellen Sie für jede Methode in Ihrer Anwendung einen Test. Sie können jede Methode in den Klassen Ihrer Anwendung testen, um sicherzustellen, dass die erwartete Operation durchgeführt wird.

Mit dem Testprojekt-Experten können Sie eine Testprojektdatei generieren, die eine .NET-Assemblierung erzeugt. Diese Assemblierung muss mit einem der NUnit-Test-Runner ausgeführt werden. Mit dem Testfall-Experten generieren Sie eine Skeleton-Testmethode für jede Methode der zu testenden Klasse. Es werden auch lokale Variablen-deklarationen für die Parameter der aufgerufenen Methode erstellt. Sie müssen den Quelltext zum Einrichten der Parameter für den Aufruf und den Quelltext zur Überprüfung der Rückgabewerte oder anderer Statusinformationen nach dem Methodenaufruf erstellen.

Das folgende Beispiel zeigt ein kleines C#-Programm, in dem eine einfache Addition und Subtraktion durchgeführt wird:

```
using System;

namespace CSharpCalcLib
{
    /// <summary>
    /// Einfache Rechnerbibliothek
    /// </summary>
    public class Calc
    {
        public int Add(int x, int y)
        {
            return x + y;
        }

        public int Sub(int x, int y)
        {
            return x - y;
        }
    }
}
```

```
    }
```

Das folgende Beispiel zeigt die Testfall-Skeleton-Datei, die Sie ändern müssen, um die zwei Methoden [Add](#) und [Sub](#) zu testen.

```
namespace TestCalc
{
    using System;
    using System.Collections;
    using System.ComponentModel;
    using System.Data;
    using NUnit.Framework;
    using CSharpCalcLib;

    // Testmethoden für Klasse TCalc
    [TestFixture]
    public class TestCalc
    {

        private Calc aCalc;

        [SetUp]
        public void SetUp()
        {
            aCalc = new Calc();
        }

        [TearDown]
        public void TearDown()
        {
            aCalc = null;
        }

        [Test]
        public void TestAdd()
        {
            int x;
            int y;
            int returnValue;
            // TODO: Aufrufparameter einrichten
            returnValue = aCalc.Add(x, y);
            // TODO: Rückgabewert validieren
        }

        [Test]
        public void TestSub()
        {
            int x;
            int y;
            int returnValue;
            // TODO: Aufrufparameter einrichten
            returnValue = aCalc.Sub(x, y);
            // TODO: Rückgabewert validieren
        }
    }
}
```

Anmerkung: Jede Testmethode wird in C#-Projekten automatisch mit dem Attribut [\[Test\]](#) versehen. Außerdem werden die Testmethoden in C# als Funktionen mit dem Rückgabewert **void** definiert.

Das folgende Beispiel zeigt ein kleines Delphi für .NET-Programm, in dem eine einfache Addition und Subtraktion durchgeführt wird:

```
unit CalcUnit;
```

```
// .Net-Version

interface

type
{ TCalc }

TCalc = class
public
  function Add(x, y: Integer): Integer;
  function Sub(x, y: Integer): Integer;
end;

implementation

{ TCalc }

function TCalc.Add(x, y: Integer): Integer;
begin
  Result := x + y;
end;

function TCalc.Sub(X, Y: Integer): Integer;
begin
  Result := x + y;
end;

end.
```

Das folgende Beispiel zeigt die Testfall-Skeleton-Datei, die Sie ändern müssen, um die zwei Funktionen `Add` und `Sub` zu testen.

```
unit TestCalcUnit;

interface

uses
  NUnit.Framework, CalcUnit;

type
  // Testmethoden für Klasse TCalc
  [TestFixture]
  TestTCalc = class
  strict private
    FCalc: TCalc;
  public
    [SetUp]
    procedure SetUp;
    [TearDown]
    procedure TearDown;
  published
    [Test]
    procedure TestAdd;
    [Test]
    procedure TestSub;
  end;

implementation

procedure TestTCalc.SetUp;
begin
  FCalc := TCalc.Create;
end;

procedure TestTCalc.TearDown;
begin
  FCalc := nil;
```

```

end;

procedure TestTCalc.TestAdd;
var
  ReturnValue: Integer;
  y: Integer;
  x: Integer;
begin
  // TODO: Aufrufparameter einrichten
  ReturnValue := FCalc.Add(x, y);
  // TODO: Rückgabewert validieren
end;

procedure TestTCalc.TestSub;
var
  ReturnValue: Integer;
  y: Integer;
  x: Integer;
begin
  // TODO: Aufrufparameter einrichten
  ReturnValue := FCalc.Sub(x, y);
  // TODO: Rückgabewert validieren
end;

end.

```

Anmerkung: In Delphi für .NET sind die Testmethoden als Prozeduren definiert.

Jede Testmethode muss folgende Bedingungen erfüllen:

- Sie muss als **public** definiert werden.
- Sie muss in Delphi für .NET eine Prozedur oder in C# eine Funktion mit dem Rückgabewert **void** sein.
- Sie darf keine Argumente haben.

Setup

Verwenden Sie die Prozedur `SetUp`, um die Variablen zu initialisieren oder Ihre Tests anderweitig vor der Ausführung vorzubereiten. Hier können Sie beispielsweise eine Datenbankverbindung einrichten, falls diese im Test benötigt wird.

TearDown

Verwenden Sie die Methode `TearDown`, um die Variablenzuweisungen zu bereinigen, den Speicher freizugeben oder andere Operationen mit Ihren Tests durchzuführen. Hier können Sie beispielsweise eine Datenbankverbindung schließen.

NUnit-Annahmen

NUnit verfügt über verschiedene Annahmen, die Sie in Ihren Tests verwenden können.

Funktion	Beschreibung	Syntax
AreEqual	Prüft, ob zwei Elemente gleich sind.	<code>Assert.AreEqual(expected, actual [, string message])</code>
IsNull	Prüft, ob ein Element Null ist.	<code>Assert.IsNull(object [, string message])</code>
IsNotNull	Prüft, ob ein Element ungleich Null ist.	<code>Assert.IsNotNull(object [, string message])</code>
AreSame	Prüft, ob zwei Elemente gleich sind.	<code>Assert.AreSame(expected, actual [, string message])</code>
IsTrue	Prüft, ob ein Element True ist.	<code>Assert.IsTrue(bool condition [, string message])</code>

IsFalse	Prüft, ob ein Element False ist.	Assert.IsFalse(bool condition [, string message])
Fail	Lässt den Test fehlschlagen.	Assert.Fail([string message])

Sie können mehrere Annahmen in einer Testmethode verwenden. Mit diesen Annahmen können Sie testen, ob eine Methode die erwartete Operation ausführt. Wenn eine Annahme nicht zutrifft, schlägt die gesamte Testmethode fehl, und die restlichen Annahmen in der Methode werden ignoriert. Nachdem Sie den Fehler behoben und die Tests erneut ausgeführt haben, werden die anderen Annahmen ausgeführt, sofern keine davon fehlschlägt.

NUnit-Test-Runner

Mit einem Testprogramm (Runner) können Sie die Tests unabhängig von Ihrer Anwendung ausführen. Die NUnit-Test-Runner laden eine Assemblierung, die die Unit-Tests enthält. Die Test-Runner identifizieren die Tests in der Assemblierung anhand der Attribute **[TEST]**, die vom Testfall-Experten erzeugt werden.

NUnit enthält zwei ausführbare Test-Runner-Dateien:

- **NUnitConsole.exe** — Der *Konsolen-Test-Runner*, ein textbasierter Test-Runner, der die Testausgabe an die Konsole sendet.
- **NUnitGUI.exe** — Der *GUI Test-Runner*, ein interaktiver GUI-basierter Test-Runner, der mit Farbcodierungen Erfolg oder Fehlschlag kennzeichnet.

Der GUI Test-Runner ist besonders hilfreich, wenn Sie interaktiv Unit-Tests oder den zu testenden Quelltext entwickeln. Die erfolgreichen Tests werden mit grün, die fehlgeschlagenen Tests mit rot und die ausgelassenen Tests mit gelb gekennzeichnet.

Verwenden Sie den Konsolen-Test-Runner, wenn Sie fertig gestellten Quelltext testen oder die Tests über automatisierte Build-Scripts ausführen. Wenn das Testergebnis in eine Datei geschrieben werden soll, übergeben Sie den Parameter **redirection**. Das folgende Beispiel zeigt, wie das Testergebnis in die Textdatei **TestResult.txt** umgeleitet wird:

```
nunit-console nunit.tests.dll /out:TestResult.txt
```

Anmerkung: Sie müssen möglicherweise den Pfad der Host-Anwendung im Dialogfeld Projektoptionen eingeben. Geben Sie mit der Eigenschaft Host-Anwendung den Pfad des gewünschten Testprogramms an.

Siehe auch

Überblick über das Testen von Units (siehe Seite 1425)

Überblick über DUnit (siehe Seite 1429)

Testfälle erstellen (siehe Seite 1427)

[NUnit.org Dokumentation](#)

10.6 Anwendungen lokalisieren

Zu RAD Studio gehören verschiedene Übersetzungs-Tools, die Sie bei der Lokalisierung und Entwicklung von .NET- und Win32-Anwendungen in anderen Sprachen verwenden können. Folgende Übersetzungs-Tools sind verfügbar:

- Experte für Satellite-Assemblierungen (für .NET)
- Ressourcen-DLL-Experte (für Win32)
- Translation-Manager
- Übersetzungswörterbuch

Die Übersetzungs-Tools stehen für Delphi VCL Forms-Anwendungen (.NET und Win32) sowie für Win32-Konsolenanwendungen, Packages und DLLs zur Verfügung. Um auf die Konfigurationsoptionen für die Übersetzungs-Tools zuzugreifen, wählen Sie **Tools > Optionen > Optionen für Übersetzungs-Tools**.

Die Experten

Bevor Sie den Translation-Manager und das Übersetzungswörterbuch verwenden können, müssen Sie Ihrem Projekt Sprachen hinzufügen. Dies geschieht mit dem Experten für Satellite-Assemblierungen (.NET-Projekte) oder mit dem Ressourcen-DLL-Experten (Win32-Projekte). Der Experte für Satellite-Assemblierungen generiert für jede Sprache, die Sie hinzufügen, eine .NET-Satellite-Assemblierung. Der Ressourcen-DLL-Experte erstellt für jede Sprache eine Win32-Ressourcen-DLL. Der Einfachheit halber wird in dieser Dokumentationen für Satellite-Assemblierungen und Ressourcen-DLLs der Begriff *Ressourcenmodul* verwendet.

Jeder Experte ermöglicht das Hinzufügen zusätzlicher Dateien, z.B. `.resx`- oder `.rc`-Dateien, die normalerweise nicht Teil eines Projekts sind. Sie können jederzeit weitere Ressourcenmodule in Ihr Projekt aufnehmen. Sind in der IDE mehrere Projekte geöffnet, können diese (oder ein Teil davon) auf einmal verarbeitet werden.

Mit Hilfe der Experten können Sie auch Sprachen aus einem Projekt entfernen und Sprachen für ein Projekt wiederherstellen.

Translation-Manager

Nachdem Sie Ressourcenmodule zum Projekt hinzugefügt haben, können Sie mit dem Translation-Manager VCL-Formulare und Ressourcen-Strings anzeigen und bearbeiten. Wenn die Bearbeitung der Übersetzungen abgeschlossen ist, können alle Ressourcenmodule der Anwendung in einem Schritt aktualisiert werden.

Für den Einsatz außerhalb der IDE steht ein externer Translation-Manager (ETM) zu Verfügung. Dieser bietet neben den Funktionen des "normalen" Translation-Managers einige zusätzliche Menüs und Symbolleisten.

Übersetzungswörterbuch

Das Übersetzungswörterbuch ist eine Datenbank für Übersetzungen, die projektubergreifend von mehreren Entwicklern genutzt werden kann. Während der Arbeit mit dem Translation-Manager können Sie übersetzte Strings im Wörterbuch speichern bzw. aus diesem abrufen.

Bei jeder Aktualisierung Ihrer Assemblierungen wird das Wörterbuch nach passenden Strings durchsucht. Diese Strings werden dann automatisch in die Assemblierungen übernommen. Sie können aber auch direkt auf das Wörterbuch zugreifen und Strings suchen, bearbeiten und löschen.

Das Übersetzungswörterbuch enthält Daten im XML-Format. Die Datei trägt standardmäßig den Namen `default.tmx` und befindet sich im Verzeichnis `RAD Studio\bin`.

Von Übersetzungs-Tools erzeugte Dateien

Die folgenden Dateien werden von den Übersetzungs-Tools generiert:

Dateierweiterungen	Beschreibung
.nfn (.NET) .dfn (Win32)	Für jedes Formular in der Anwendung und für jede Zielsprache wird eine separate Datei verwaltet. Diese Dateien enthalten die Daten (einschließlich der übersetzten Strings), die im Translation-Manager angezeigt werden.
.resx (.NET)	Der Experte für Satellite-Assemblierungen erzeugt aus der vom Compiler generierten .drcil-Datei für jede Zielsprache eine .resx-Datei. Die .resx-Dateien enthalten spezielle Kommentare, die von den Übersetzungs-Tools verwendet werden.
.rc (Win32)	Der Ressourcen-DLL-Experte erzeugt aus der vom Compiler generierten .drc-Datei für jede Zielsprache eine .resx-Datei. Die .resx-Dateien enthalten spezielle Kommentare, die von den Übersetzungs-Tools verwendet werden.
.tmx	Die Daten des Übersetzungswörterbuchs werden in einer .tmx-Datei gespeichert. Bei Bedarf können mehrere .tmx-Dateien für unterschiedliche Wörterbücher verwaltet werden.
.bdsproj	Der externe Translation-Manager speichert die Assemblierungen (Sprachen) und die zu übersetzenden Ressourcen in einer .bdsproj-Projektdatei. Wenn externe Übersetzer Sprachen zu einem Projekt hinzufügen oder daraus entfernen, speichern sie die Änderungen in einer .bdsproj-Datei und übergeben diese an den Entwickler.

Anmerkung: Die genannten Dateien dürfen nicht manuell bearbeitet werden.

Siehe auch

Sprachen zu einem Projekt hinzufügen ([siehe Seite 86](#))

Ressourcenmodule aktualisieren ([siehe Seite 93](#))

Ressourcendateien im Translation-Manager bearbeiten ([siehe Seite 88](#))

Aktive Sprache für ein Projekt festlegen ([siehe Seite 88](#))

Externen Translation-Manager einrichten ([siehe Seite 91](#))

10.7 Anwendungen debuggen

In den meisten Umgebungen werden zum Debuggen identische oder ähnliche Techniken benutzt. RAD Studio bietet eine integrierte Debugging-Umgebung für die Fehlersuche in Win32- und .NET-Anwendungen. Der Debugger kann auch zum Debuggen von Anwendungen auf einem externen Computer eingesetzt werden, auf dem RAD Studio nicht installiert ist.

10.7.1 Überblick zum Debuggen

In RAD Studio steht sowohl der CodeGear .NET-Debugger als auch der CodeGear Win32-Debugger zur Verfügung. Die IDE verwendet automatisch den richtigen Debugger für den aktiven Projekttyp. Das plattformübergreifende Debuggen innerhalb einer Projektgruppe wird unterstützt. Die Benutzeroberfläche der beiden Debugger ist größtenteils identisch.

Mit Hilfe der integrierten Debugger lassen sich sowohl Laufzeitfehler als auch Fehler in der Logik einer RAD Studio-Anwendung finden. Bei der Arbeit mit den Debuggern können Sie den Quelltext schrittweise durchlaufen, Haltepunkte und überwachte Ausdrücke festlegen sowie Programmwerte prüfen und bearbeiten. Wenn Sie eine Anwendung debuggen, unterstützen Sie die Debug-Fenster in einer Sitzung bei der Fehlersuche. Sie zeigen Informationen über den Status Ihrer Anwendung an.

Fehlersuche im Quelltext schrittweise ausführen

Wenn Sie Ihren Quelltext beim Debuggen schrittweise durchlaufen lassen, führen Sie das Programm Codezeile für Codezeile aus. Sie können bei jedem Schritt den Programmstatus überprüfen, die Programmausgabe betrachten, Datenwerte des Programms ändern und dann mit der Ausführung der nächsten Codezeile fortfahren. Die nächste Codezeile wird erst dann ausgeführt, wenn Sie den Debugger dazu anweisen.

Im Menü Start stehen die Befehle Einzelne Anweisung und Gesamte Routine zur Auswahl. Beide Befehle weisen den Debugger an, die nächste Codezeile auszuführen. Enthält die Zeile jedoch einen Funktionsaufruf, führt der Befehl Einzelne Anweisung die Funktion aus und hält an der ersten Codezeile *innerhalb* dieser Funktion an. Der Befehl Gesamte Routine führt die Funktion aus und hält an der ersten Zeile *nach* der Funktion an.

Auswerten/Ändern

Mit der Funktion **Auswerten/Ändern** können Sie Ausdrücke auswerten und Variablenwerte ändern. Die Funktionalität von **Auswerten/Ändern** variiert je nach verwendeter Sprache. Bei einem C++ Projekt können im Dialogfeld **Auswerten/Ändern** nur C++ Ausdrücke eingegeben werden. Bei einem C#-Projekt werden im Dialogfeld **Auswerten/Ändern** nur C#-Ausdrücke akzeptiert. Bei Delphi-Projekten sind im Dialogfeld **Auswerten/Ändern** nur Delphi-Ausdrücke zulässig.

Haltepunkte

Haltepunkte unterbrechen die Programmausführung an einem bestimmten Punkt im Programm oder bei Auftreten einer bestimmten Bedingung. Sie können dann mit dem Debugger den Status Ihres Programms untersuchen oder die gesamte Routine oder einzelne Anweisungen ausführen. Der Debugger unterstützt vier Arten von Haltepunkten.

- Quelltexthaltepunkte unterbrechen die Ausführung an einer bestimmten Position im Quelltext.
- Adresshaltepunkte unterbrechen die Ausführung an einer bestimmten Speicheradresse.
- Datenhaltepunkte unterbrechen die Ausführung, wenn sich Speicher an einer bestimmten Adresse ändert.
- Haltepunkte für das Laden eines Moduls unterbrechen die Ausführung, wenn das angegebene Modul geladen wird.

Anmerkung: Datenhaltepunkte werden nur im Win32-Debugger unterstützt.

Überwachte Ausdrücke

Ein überwachter Ausdruck ermöglicht Ihnen, die Werte von Programmvariablen oder -ausdrücken zu verfolgen, während Sie mit dem Befehl Gesamte Routine oder Einzelne Anweisung durch den Quelltext gehen. Während das Programm im Debugger läuft, ändert sich der Wert des überwachten Ausdrucks, sobald das Programm die Variablen der überwachten Ausdrücke aktualisiert.

Fenster zur Fehlersuche

Die folgenden Debug-Fenster unterstützen Sie bei der Fehlersuche in einem Programm. Die meisten dieser Fenster werden automatisch angezeigt, wenn Sie eine Debug-Sitzung beginnen. Sie können die Fenster aber auch einzeln anzeigen, indem Sie das Untermenü **Ansicht**►**Debug-Fenster** zur Auswahl verwenden.

In jedem Fenster stehen ein oder mehrere Kontextmenüs zur Auswahl, wenn Sie mit der rechten Maustaste in das Fenster klicken. Mit der Hilfe-Taste F1 können Sie für jedes Fenster detaillierte Informationen sowohl zu dem Fenster als auch zu den Kontextmenüs anzeigen.

Fenster zur Fehlersuche	Beschreibung
Liste der Haltepunkte	Zeigt alle aktuell im Quelltext-Editor oder im Fenster CPU gesetzten Haltepunkte an.
Aufruf-Stack	Zeigt die aktuelle Sequenz der Funktionsaufrufe an.
Liste überwachter Ausdrücke	Zeigt den aktuellen Wert überwachter Ausdrücke im Bereich des Ausführungspunkts an.
Lokale Variablen	Zeigt die lokalen Variablen der aktuellen Funktion an. Damit lässt sich überwachen, wie das Programm Variablenwerte während der Programmausführung aktualisiert.
Module	Zeigt Prozesse unter der Kontrolle des Debuggers und die aktuell von jedem Prozess geladenen Module an. Darüber hinaus wird eine hierarchische Darstellung der in der Anwendung benutzten Namespaces, Klassen und Methoden angezeigt.
Thread-Status	Zeigt den Status aller Prozesse und Ausführungs-Threads an, die beim Debuggen in einer Anwendung ausgeführt werden. Dies ist besonders beim Debuggen von Multithread-Anwendungen hilfreich.
Ereignisprotokoll	Zeigt Meldungen zur Prozess-Steuerung, zu Haltepunkten, Ausgabe, Threads und Modulen an.
CPU	Zeigt den Status eines Programms auf niedrigster Ebene an, einschließlich der Assemblierungsanweisungen für jede Zeile des Quelltextes und des Inhalts bestimmter Register.
FPU	Zeigt den Inhalt der Gleitkommeinheit und der SSE-Register des Prozessors an.

Externe Fehlersuche

Mit dem externen Debugger können Sie eine Anwendung debuggen, die sich auf einem externen Computer befindet. Ihr Computer muss über eine TCP/IP-Verbindung mit dem externen Computer verbunden sein. Außerdem muss der externe Debugger auf dem externen Computer installiert sein. Nachdem die erforderlichen Anwendungsdateien erstellt und auf den externen Computer kopiert wurden, können Sie eine Verbindung zu diesem Computer herstellen und mit dem Debuggen beginnen.

Siehe auch

Überblick zum externen Debugger (siehe Seite 1441)

Das Projekt zum Debuggen vorbereiten (siehe Seite 27)

Quelltexthaltepunkte setzen und bearbeiten (siehe Seite 19)

Ausdruck hinzufügen (siehe Seite 17)

Variablenausdrücke bearbeiten (siehe Seite 26)

Werte von Datenelementen untersuchen und ändern (siehe Seite 24)

Debuggen externer Anwendungen (siehe Seite 32)

Mit einem ausgeführten Prozess verbinden (siehe Seite 18)

10.7.2 Überblick zum externen Debugger

Sie können über die IDE auf Ihrem lokalen Computer Anwendungen debuggen, die sich auf einem externen Computer befinden. Dieses Vorgehen ist dann von Vorteil, wenn die Installation der IDE und die Neuerstellung eines Projekts auf einem Computer nicht möglich oder zu aufwändig ist. Das externe Debuggen ist auch bei Anwendungen sinnvoll, die auf den Rechnern der Endbenutzer anders ausgeführt werden als auf Ihrem lokalen Rechner.

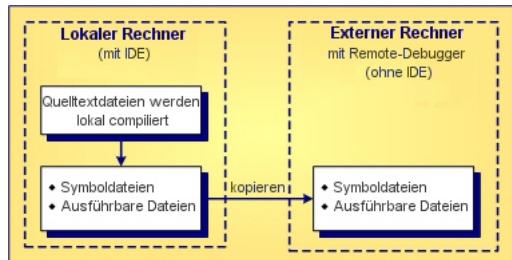
Der externe Debugger

Die ausführbare Datei des externen Debuggers hat den Namen rmtdbg105.exe. Diese Datei sowie alle unterstützenden Dateien müssen auf dem externen Computer vorhanden sein. Am einfachsten lässt sich der externe Debugger direkt vom RAD Studio-Installationsdatenträger installieren. Steht dieser nicht zur Verfügung, erstellen Sie auf einem Computer, auf dem die vollständige RAD Studio-IDE installiert ist, Kopien der erforderlichen Dateien.

Lokale und externe Dateien

Beim externen Debuggen sind drei Arten von Dateien beteiligt:

- Quelldateien
- Ausführbare Dateien
- Symboldateien



Quelldateien werden in der IDE auf dem lokalen Computer kompiliert. Ausführbare Dateien und die Symboldateien, die nach der Kompilierung erstellt werden, müssen auf den externen Computer kopiert werden.

Quelldateien

Beim Debuggen eines Projekts auf einem externen Rechner müssen die Quelldateien für das Projekt auf dem lokalen Computer geöffnet sein. Die Quelldateien werden im Editorfenster angezeigt, sodass der aktuelle Ausführungspunkt des Programms erkennbar ist. Die Quelldateien werden auf dem externen Computer nicht benötigt.

Ausführbare Dateien

Unter ausführbaren Dateien versteht man die DLL- und EXE-Dateien im Adressraum der Anwendung. Diese Dateien werden auf dem lokalen Computer erstellt und dann auf den externen Rechner kopiert.

Symboldateien

Symboldateien werden während des Compilierens auf dem lokalen Computer erstellt. Der Debugger ermittelt anhand dieser Dateien verschiedene Informationen, z.B. die Zuordnung von Maschinenanweisungen zu Quelltextzeilennummern oder die

Namen und Typen von Variablen, die in den Quelldateien deklariert sind. Die Erweiterung der Symboldateien variiert je nach Sprache (siehe die folgende Tabelle).

Sprache	Erweiterung der Debug-Symboldateien
Delphi für Win32	.rsm
Delphi für .NET	.rsm und .pdb
C++	.tds
C#	.pdb

Für die Erstellung der Symboldateien auf dem lokalen Computer müssen spezielle Einstellungen festgelegt werden. Nach der Erstellung können Sie die Dateien auf den externen Computer kopieren.

Lokaler und externer Computer

Für das externe Debuggen müssen Sie sich beim externen Computer anmelden können. Außerdem benötigen Sie Schreibzugriff auf mindestens ein Verzeichnis.

Anmerkung: Der externe Debugger stellt keinen Mechanismus für die Interaktion mit einer Anwendung auf dem externen Computer bereit. Falls eine Interaktion erforderlich ist, müssen Sie eine Verbindung zum externen Desktop einrichten.

Siehe auch

Externe Anwendung debuggen ([siehe Seite 32](#))

Debugger auf einem externen Computer installieren ([siehe Seite 30](#))

Verbindung für das externe Debuggen einrichten ([siehe Seite 33](#))

Dateien für das externe Debuggen vorbereiten ([siehe Seite 35](#))

10.8 Deployment von Anwendungen

Nachdem Sie eine Anwendung erstellt, getestet und auf Fehler überprüft haben, können Sie sie an andere weitergeben. Je nach Größe und Komplexität einer Anwendung sollten Sie diese als eine oder mehrere Assemblierungen, als komprimierte Cabinet-Dateien ([.CAB](#)) oder im Format eines Installationsprogramms (zum Beispiel [.MSI](#)) packen. Nachdem die Anwendung gepackt ist, können Sie sie mit XCOPY, FTP, als Download oder mit einem Installationsprogramm verteilen.

In diesem Abschnitt werden folgende allgemeine Themen behandelt:

- Deployment von .NET-Anwendungen
- Deployment von Win32-Anwendungen
- Installationsprogramme verwenden
- RAD Studio-Dateien weitergeben
- Software von Drittanbietern weitergeben

Weitere Informationen zum Deployment spezieller Anwendungstypen erhalten Sie über die Links am Ende dieses Themas.

Deployment von .NET-Anwendungen

Wenn auf dem Zielcomputer bereits das .NET Framework installiert ist, umfasst das Deployment einer einfachen Anwendung, die aus einer einzelnen ausführbaren Datei besteht, nur das Kopieren der [.EXE](#)-Datei auf den Zielcomputer. Die Anwendung muss in diesem Fall nicht registriert werden, und ein einfaches Löschen der Anwendungsdateien deinstalliert diese Anwendung wieder.

Anwendungen mit gemeinsamen Assemblierungen

Wenn Ihre Anwendung eine Assemblierung enthält, die auch von anderen Anwendungen genutzt wird, müssen Sie die Assemblierung mit einem starken Namen versehen, um sie eindeutig identifizierbar zu machen. Dann installieren Sie die Assemblierung im Globalen Assemblierungs-Cache (GAC). Ein starker Name besteht aus dem Textnamen der Assemblierung, der Versionsnummer, optionalen Kulturinformationen sowie dem öffentlichen Schlüssel und der digitalen Signatur, die ihre Eindeutigkeit gewährleisten. Im .NET Framework SDK stehen Befehlszeilen-Utilities zum Erstellen öffentlicher und privater Schlüssel ([SN.EXE](#)) sowie zum Zuweisen starker Namen ([AL.EXE](#)) und zur Installation einer Assemblierung im GAC ([GACUTIL.EXE](#)) bereit. Weitere Informationen zu diesen Utilities finden Sie in der Online-Hilfe zum Framework SDK.

Deployment von VCL.NET-Anwendungen

Bei Anwendungen, die das VCL.NET-Framework verwenden, bestimmt die Art der Erstellung, welche Dateien weitergegeben müssen. Wenn VCL für .NET-Units direkt in die ausführbare Datei des Programms kompiliert werden, sind externe Abhängigkeiten nur bezüglich des .NET-Framework vorhanden.

Compiler Sie die Anwendung jedoch mit externen Referenzen auf VCL-für-.NET-Assemblierungen, sind externe Referenzen auf das .NET Framework, auf [Borland.Delphi.dll](#) und auf alle im Projekt verwendeten RAD Studio-Packages (z.B. [Borland.VclRtl.dll](#) oder [Borland.Vcl.dll](#)) vorhanden.

Deployment von ASP.NET-Anwendungen

Zu RAD Studio gehört der ASP.NET-Deploymentmanager, der Sie beim Deployment Ihrer ASP.NET-Anwendungen unterstützt. Mit seiner Hilfe können Sie das Deployment auf einem externen Computer (mittels einer Freigabe oder einer FTP-Verbindung) oder auf Ihrem lokalen Computer durchführen. Wenn Sie für Ihr Projekt einen Deploymentmanager bereitstellen, wird eine XML-Datei ([.bdsdeploy](#)) in das Projektverzeichnis eingefügt, und in der IDE wird eine eigene Registerkarte für das Deployment angezeigt. Auf dieser Registerkarte geben Sie die erforderlichen Informationen für das Ziel und die Verbindung ein.

Sie können bei Bedarf die Liste der zu kopierenden Dateien modifizieren. Der Deploymentmanager kopiert die Dateien dann zum angegebenen Ziel.

Weitergeben des .NET Framework

Wenn Sie Ihre Anwendung an einen Computer weitergeben möchten, auf dem das .NET Framework nicht installiert ist, müssen Sie auch das .NET Framework weitergeben und dieses zusammen mit Ihrer Anwendung installieren. Microsoft stellt dafür ein Installationsprogramm namens [DOTNETFX.EXE](#) bereit, das die allgemeine Laufzeitumgebung und diejenigen Komponenten des .NET Framework installiert, die zum Ausführen von .NET-Anwendungen erforderlich sind. Weitere Informationen zu [DOTNETFX.EXE](#) finden Sie in der Online-Hilfe des .NET Framework SDK.

Vorbereitungen für das Deployment einer C#-Anwendung

Bei der Entwicklung einer C#-Anwendung compilieren Sie diese üblicherweise bereits mit Debug-Informationen, um das Testen zu vereinfachen. Wenn Sie ein neues Projekt erstellen, ist dafür standardmäßig die Option **Fehlersuche** eingestellt, wodurch die ausführbaren Dateien und die Programmdatenbankdatei ([.PDB](#)) für das Debuggen im Verzeichnis `projekt\BIN\DEBUG` erstellt werden.

Wenn die C#-Anwendung für das Deployment bereit ist, können Sie sie mit der Standardoption oder einer benutzerdefinierten **Ausgabe**-Option compilieren, um eine optimierte Version der Anwendung im Verzeichnis `projekt\BIN\RELEASE` zu erstellen. Eine optimierte Anwendung ist kleiner, schneller und effizienter. Um die Einstellungen der Optionen **Fehlersuche/Ausgabe** zu ändern, wählen Sie [Projekt>Optionen](#).

Deployment von Win32-Anwendungen

Informationen zum Deployment von Win32-Anwendungen erhalten Sie über den Link [Deployment von Win32-Anwendungen](#) am Ende dieses Themas.

Installationsprogramme verwenden

Für komplexe Anwendungen, die aus mehreren Dateien bestehen, können Sie ein Installationsprogramm verwenden. Installationsprogramme führen verschiedene Operationen durch, wie beispielsweise Kopieren der ausführbaren und unterstützenden Dateien auf den Zielcomputer und Vornehmen der Windows-Registrierungseinträge.

Setup-Toolkits wie InstallShield Express automatisieren die Erstellung von Installationsprogrammen. Zusätzlicher Quelltext ist in den meisten Fällen nicht erforderlich. InstallShield Express basiert auf der MSI-Technologie (Windows Installer) und kann von der RAD Studio-Installations-CD installiert werden. Nach der Installation finden Sie weitere Informationen zur Verwendung von InstallShield in der Online-Hilfe.

RAD Studio-Dateien weitergeben

Viele der Dateien, die mit RAD Studio-Anwendungen verbunden sind, unterliegen bei der weiteren Verteilung bestimmten Beschränkungen oder dürfen überhaupt nicht verteilt werden. In den folgenden Dokumenten finden Sie die rechtlichen Beschränkungen in Bezug auf die Weitergabe dieser Dateien.

Datei	Beschreibung
deploy.htm	Enthält Erläuterungen zum Deployment für jede Edition von RAD Studio.
license.txt	Enthält die rechtlichen Vereinbarungen und Verpflichtungen bezüglich der Verwendung von RAD Studio.
readme.htm	Diese Datei enthält "Informationen in letzter Minute" zu RAD Studio. Diese sind zwar meist technischer Natur, können aber auch die Rechte bei der Weitergabe von RAD Studio-Dateien betreffen.

Diese Dateien finden Sie im Verzeichnis [C:\Programme\CodeGear\RAD Studio\4.0](#).

Software von Drittanbietern weitergeben

Die Rechte zur Weitergabe der Software – z.B. Komponenten, Utilities und Hilfsprogramme – anderer Hersteller, werden durch

die jeweiligen Hersteller dieser Software geregelt. Bevor Sie die Software eines Drittanbieters zusammen mit Ihrer RAD Studio-Anwendung weitergeben, informieren Sie sich bitte bei dem Hersteller oder in der Softwaredokumentation über die diesbezüglichen Rechte.

Siehe auch

[Deployment von Datenbankanwendungen für das .NET Framework](#)

[Deployment von ASP.NET-Anwendungen](#)

[Deployment von COM Interop-Anwendungen](#)

[Deployment von ECO-aktivierten Anwendungen](#)

[Deployment von Win32-Anwendungen](#)

[Microsoft – Übersicht zum Deployment von Anwendungen](#)

10.9 Anwendungen mit Together modellieren

Dieser Abschnitt gibt einen Überblick über die Funktionen und Leistungsmerkmale von **Borland Together**.

10.9.1 Überblick zur Modellierung

Durch eine effektive Modellierung in Together wird die Entwicklungsphase eines **Projekts** wesentlich vereinfacht. Die nahtlose Integration in RAD Studio ermöglicht eine problemlose Umsetzung des **Modells** in Quelltext.

Wichtigstes Ziel der Modellierung ist es, die Struktur und die Komponenten eines Softwaresystems zu organisieren und offen zu legen. Anforderungen, Subsysteme, logische und physische Elemente sowie Struktur- und Verhaltensmuster werden im **Modell** grafisch dargestellt.

Die Vorteile der Modellierung für die Softwareentwicklung sind heute den meisten Programmierern bekannt. Together bietet zusätzlich zu diesen Vorteilen eine volle Synchronisierung von **Diagramm** und Quelltext.

Siehe auch

Allgemeines zu Together (↗ siehe Seite 1380)

Together-Projekte – Überblick (↗ siehe Seite 1446)

Together-Diagramme – Anleitungen (↗ siehe Seite 1395)

10.9.2 Together-Projekte – Überblick

Die gesamte Arbeit in Together erfolgt immer im Kontext eines **Projekts**. Ein Projekt ist eine logische Struktur mit allen Ressourcen, die Sie für Ihre Arbeit benötigen. In Together gibt es folgende Arten von Projekten: **Designprojekte** und **Implementierungsprojekte**. Außerdem stehen mehrere Projektformate zur Verfügung.

Sie müssen selbst festlegen, welche Verzeichnisse, Archive und Dateien in ein Projekt aufgenommen werden. Die Projekteigenschaften können bei der Erstellung des Projekts konfiguriert und später im Objektinspektor geändert werden.

Siehe auch

UML 2.0-Beispielprojekt (↗ siehe Seite 1384)

Ein Projekt erstellen (↗ siehe Seite 249)

Unterstützte Projektformate (↗ siehe Seite 1332)

10.9.3 Überblick zu Namespaces und Paketen

Ein **Namespace** ist ein Element in einem Modell, das eine Gruppe benannter Elemente enthält, die über ihren Namen identifiziert werden kann.

Ein Projekt besteht aus einem oder mehreren Namespaces (Paketen). Die Begriffe "Namespace" und "Paket" sind praktisch gleichbedeutend. "Namespace" wird für Implementierungsprojekte verwendet, "Paket" für Designprojekte.

Ein Namespace (Paket) lässt sich mit einem Container für Diagramme und Modellelemente vergleichen. Der Inhalt eines Namespaces (Pakets) kann in einer speziellen Art von Klassendiagramm dargestellt werden.

Jedes Projekt enthält nach seiner Erstellung den Standard-Namespace (das Standardpaket) **default**.

Siehe auch

Mit Namespaces und Paketen arbeiten (siehe Seite 225)

Klassendiagrammtypen (siehe Seite 1295)

10.9.4 Überblick über Together-Diagramme

Diagramme können als Graphen mit Eckpunkten und Kanten betrachtet werden, die basierend auf einem bestimmten **Algorithmus** angeordnet sind.

Jedes Diagramm gehört zu einem **Diagrammtyp** (z.B. UML 2.0-Klassendiagramm). Der Diagrammtyp bestimmt, welcher Satz von **Modellelementen** für das Diagramm verfügbar ist.

Diagramme existieren innerhalb des Kontexts eines Namespace (Pakets). Bevor Sie ein neues Diagramm erstellen können, müssen Sie ein Projekt oder eine Projektgruppe anlegen. Wenn die Unterstützung für Together aktiviert ist, wird standardmäßig ein Paketdiagramm auf Projektebene erzeugt. Sie können in dem Projekt verschiedene UML-Diagramme erstellen.

Zusätzlich zu den Standardeigenschaften von Diagrammen und Diagrammelementen können Sie **Benutzereigenschaften** erstellen, die aus einem Name-Wert-Paar bestehen.

Siehe auch

Diagrammformat (siehe Seite 1449)

Diagramme erstellen (siehe Seite 116)

Benutzereigenschaften verwenden (siehe Seite 135)

10.9.5 Unterstützte UML-Spezifikationen

Die von der Object Management Group entwickelte Unified Modeling Language (UML) ist eine diagrammbasierte Sprache für die Visualisierung, Definition, Konstruktion und Dokumentation der Artefakte von verteilten Objektsystemen.

Together unterstützt UML und erleichtert Ihnen so das Definieren, Visualisieren und Dokumentieren der Modelle Ihrer Softwaresysteme sowie deren Struktur und Design.

Ausführliche Informationen zur UML-Semantik und -Notation finden Sie in der UML-Dokumentation. Im Dokument *UML (Version): Superstructure* sind die Konstrukte auf Benutzerebene spezifiziert, die für UML erforderlich sind. Es wird ergänzt durch das Dokument *UML (Version): Infrastructure*, in dem die grundlegenden Sprachkonstrukte für UML aufgeführt sind. Die beiden Dokumente enthalten die komplette Spezifikation für die Modellierungssprache UML.

UML 1.5 und UML 2.0

Der Typ des Projekts bestimmt, welche Diagramme verwendet werden können.

Für Designprojekte werden UML 1.5 und UML 2.0 unterstützt.

Für Implementierungsprojekte wird nur UML 1.5 unterstützt.

Die UML-Version wird beim Erstellen eines Projekts ausgewählt. Sie kann später nicht mehr geändert werden.

UML in Farbe

UML In Farbe ist ein optionales Profil, das eine **Modellierung in Farbe** ermöglicht. Durch die Modellierung in Farbe können Sie einen Problembereich analysieren und bestimmte Klassen während der Analyse einfacher erkennen. Together unterstützt vier Hauptgruppen von Stereotypen zur Farbmodellierung:

- Role
- MomentInterval, Mi-detail
- Party, Place, Thing
- Description

Sie können für jeden dieser Stereotypen eine bestimmte Farbe auswählen, um Ihr Modell besser verständlich zu gestalten. Den anderen Stereotypen können keine Farben zugewiesen werden.

Die Modellierung in Farbe wird eingehend im Buch *Java Modeling in Color with UML: Enterprise Components and Process* von Coad, Lefebvre und De Luca beschrieben.

Siehe auch

<http://www.uml.org/>

Optionen für das Diagramm-Layout (siehe Seite 1282)

10.9.6 Übersicht zu Modellelementen

Modellelemente ist der Sammelbegriff für alle Komponenten des Modells, die in ein Diagramm eingefügt werden können.

Modellelemente bestehen aus **Knoten**, die über **Beziehungen** verbunden sind.

Der Diagrammtyp bestimmt, welche Modellelemente verfügbar sind. Die verfügbaren Modellelemente werden in der Tool-Palette angezeigt.

Eine Beziehung kann über eine Beschriftung verfügen. Sie können diese Beschriftung beliebig entlang der Beziehungslinie verschieben.

Siehe auch

Einzelnes Modellelement erstellen (siehe Seite 142)

Tool-Palette (siehe Seite 1267)

10.9.7 Überblick zu Hinweisen in Diagrammen

Die Tool-Palette für UML-Diagrammelemente enthält Hinweis- und Hinweisbeziehungsschaltflächen für alle UML-Diagramme. Mit diesen Elementen können Sie Hinweisknoten und die zugehörigen Beziehungen in das Diagramm einfügen.

Hinweise können als frei platzierbare Elemente eingefügt werden. Sie haben aber auch die Möglichkeit, eine Hinweisbeziehung zu einem anderen Element zu erstellen, um zu signalisieren, dass sich dieser Hinweis speziell auf dieses Element bezieht.

Es ist möglich, eine Hinweisbeziehung mit einer anderen Beziehung zu verknüpfen.

Der Text von Hinweisen für Klassendiagrammelemente wird nicht in den Quelltext übernommen.

Siehe auch

Diagramme mit Hinweisen versehen (siehe Seite 115)

10.9.8 Überblick zu Verknüpfungen

Eine **Verknüpfung** repräsentiert ein Knotenelement in demselben oder einem anderen Diagramm.

Verknüpfungen ermöglichen die Wiederverwendung von Elementen, das Anzeigen von Bibliotheksklassen in Diagrammen und das Darstellen der Beziehungen zwischen den Diagrammen in einem Modell.

Zu den Elementen aller Projekte in der aktuellen Projektgruppe lässt sich eine Verknüpfung erstellen. Sie können auch eine Verknüpfung zu einer inneren Klasse oder einem Interface eines anderen Klassifizierers definieren. Außerdem kann eine Verknüpfung zu einem Element über die Projektreferenzen hinzugefügt werden, einschließlich Binärdateien (.dll, .exe).

Ein kleines Verknüpfungssymbol über einem Knoten zeigt an, dass eine Verknüpfung existiert. Es ist jedoch nur vorhanden, wenn der Knoten zu einem anderen Namespace oder Paket gehört.

Markieren Sie eine Verknüpfung im Diagramm, und wählen Sie im Kontextmenü den Befehl Zu Element navigieren, um das Quellelement in der Modellansicht anzuzeigen.

Siehe auch

Verknüpfungen hinzufügen (siehe Seite 139)

10.9.9 Diagrammformat – Überblick

In Together werden Diagramme in XML-Dateien mit der Erweiterung .txv(diagramm) im Ordner ModelSupport_%PROJECTNAME%ModelSupport eines Projekts gespeichert.

Die Dateien enthalten Informationen zu Diagrammelementen (Layout, Hintergrundfarbe, Stereotypen usw.).

Die Datei (Name).txvcls enthält beispielsweise ein Klassendiagramm. Alle Modellierungsprodukte (Borland Together für Visual Studio .NET, CodeGear Together ControlCenter, CodeGear Together Architect, CodeGear Together Edition für Eclipse und CodeGear Together für JBuilder) unterstützen dasselbe Diagrammformat, sodass Diagramme produktübergreifend kompatibel sind. Sie können Diagramme kopieren und in einem Produkt verwenden, in dem sie nicht erstellt wurden.

Die Diagrammelemente sind Bestandteil der übergeordneten Standard-Paket- bzw. Namespace-Dateien (default.txaPackage). Diese Dateien enthalten alle Informationen über die Elemente und ihre Eigenschaften, während die Positionen und Abmessungen der Elemente in den Diagrammdateien gespeichert werden.

Diese Version von Together verwendet das Format mit eingebetteten Modellelementen (die als Dateimitglieder erstellt werden).

Siehe auch

Together Diagramme – Überblick (siehe Seite 1447)

Diagramme erstellen (siehe Seite 116)

10.9.10 Überblick zum Diagramm-Layout

Sie haben verschiedene Möglichkeiten, die Diagrammnotation anzupassen.

Sie können in Together automatisierte Layout-Features auf einfache oder komplexe Diagramme anwenden, um das Diagramm-Layout für die Anzeige oder den Druck zu optimieren. Die Knoten und Beziehungen in einem Diagramm werden basierend auf einen bestimmten Algorithmus angeordnet.

Es ist auch möglich, die Anordnung der Elemente manuell festzulegen.

Sie haben die Wahl zwischen folgenden Layout-Algorithmen:

- **Auto-Auswahl:** Für jeden Diagrammtyp können mehrere Algorithmen in Frage kommen. Bei Auswahl dieser Option werden die internen Informationen aller Algorithmen analysiert und dann derjenige Algorithmus ausgewählt, der am besten für den aktuellen Diagrammtyp geeignet ist. Wenn **Auto-Auswahl** aktiviert ist, gilt Folgendes: Jeder Layout-Algorithmus enthält interne Informationen über die Typen der Diagramme, die er verarbeitet, sowie über die numerischen Merkmale für die Qualität des erstellten Endlayouts, wenn der Algorithmus auf die entsprechenden Diagrammtypen angewendet wird. Für jeden Diagrammtyp können mehrere Algorithmen in Frage kommen. Wenn Sie die Option **Auto-Auswahl** aktivieren, wird anhand der internen Informationen der optimale Algorithmus für den aktuellen Diagrammtyp ausgewählt.
- **Hierarchisch:** Dieser Algorithmustyp ist für die Analyse von hierarchischen Strukturen am besten geeignet (beispielsweise für die Erkennung von Vererbungsbeziehungen). Der hierarchische Algorithmus basiert auf dem Sugiyama-Algorithmus. Dieser erstellt ein hierarchisches UML-Diagramm basierend auf den festgelegten Einstellungen.
- **Together:** Dieser Algorithmus kann auf alle Diagrammtypen angewendet werden. Er umfasst die Layout-Optionen der Version 6.1 von Together ControlCenter und Together Edition für JBuilder.
- **Baumartig:** Dieser Algorithmus zeichnet ein Baumdiagramm in einem Baumlayout. Der Algorithmus zeichnet den gegebenen Graphen in einem Baumlayout gemäß der Maximalwerte seines übergreifenden Graphen (Spanning-Tree).
- **Orthogonal:** Dieser einfache Algorithmus wird verwendet, wenn die Hierarchie keine Rolle spielt. Der orthogonale Algorithmus verwendet Heuristiken zur Verteilung von Diagrammknoten in einem Gitter.
- **Spring Embedder:** Spring Embedder sind Layout-Algorithmen, die den Eingabographen als Kräftesystem interpretieren und versuchen, für dieses System eine minimale Energiekonfiguration zu errechnen. Alle Kanten werden als gerade Linien gezeichnet. Dieser Layouttyp eignet sich vor allem für Projekte mit umfangreichem Quelltext und zahlreichen Diagrammelementen. Wenn Sie das Layout eines Graphen dem Spring Embedder-Algorithmus überlassen, betrachtet das Programm den Graphen als physikalisches Modell (mit Massen und Federn), das physikalischen Kräften ausgesetzt ist. Unnötig lange Kanten werden am stärksten verkürzt. Wenn die Knoten und Kanten ausgewogen sind, haben Sie eine geometrische Repräsentation des Graphen erreicht.

Jeder Algorithmus verfügt über einen speziellen Satz von Optionen, die in der Kategorie [Together > \(Ebene\) > Diagramm > Layout](#) des Dialogfelds Optionen definiert werden.

Siehe auch

Automatisches Layout für Diagramme (siehe Seite 127)

Optionen für das Diagramm-Layout (siehe Seite 1284)

10.9.11 Überblick zu Hyperlinks

Sie können Hyperlinks zwischen Diagrammen oder Diagrammelementen und anderen Systembereichen erstellen, um eine direkte Navigation zu diesen Bereichen zu ermöglichen.

Einsatzmöglichkeiten von Hyperlinks

Sie können Hyperlinks für folgende Zwecke verwenden:

- Verknüpfen von Diagrammen mit allgemeinen Informationen oder Übersichtsinformationen mit anderen Diagrammen, die entsprechende spezielle und detaillierte Informationen enthalten.
- Erstellen von Suchsequenzen, die in einer bestimmten Reihenfolge durch verschiedene, aber verwandte Ansichten führen. Erstellen von hierarchisch aufgebauten Suchsequenzen.
- Verknüpfen von abgeleiteten Klassen mit ihren Vorfahren, Durchsuchen von Hierarchien.
- Verknüpfen von Diagrammen oder Elementen mit Standards, Referenzdokumenten oder erzeugter Dokumentation.
- Vereinfachen der Zusammenarbeit von Teammitgliedern.

Hyperlinks können von einem vorhandenen Diagramm oder einem seiner Elemente zu einem anderen Diagramm oder Diagrammelement im Projekt erstellt werden. Es ist auch möglich, ein neues Diagramm zu erstellen und es mit dem aktuellen Diagramm per Hyperlink zu verknüpfen.

Sie können Hyperlinks auch von Diagrammen zu externen Dokumenten wie Dateien oder URLs erstellen. Die meisten Benutzer kennen Hyperlinks dieser Form für Dokumente auf einem LAN- oder Dokumentserver oder für URLs im Firmen-Intranet. Es ist aber auch möglich, Links zu Online-Informationen wie Newsgroups oder Diskussionsforen zu erstellen. Zu allen Elementen, die Online verfügbar sind, kann ein Hyperlink erstellt werden.

Arten von Hyperlinks

Folgende Ziele kommen für Hyperlinks in Frage:

- Vorhandenes Diagramm oder Diagrammelement an einer beliebigen Position in der Projektgruppe
- Neues Diagramm (wird dynamisch erstellt)
- Dokument oder Datei auf einem lokalen oder externen Speichermedium
- URL im Firmen-Intranet oder im Internet

Siehe auch

Diagramme über Hyperlinks verknüpfen (siehe Seite 125)

Abfolgen von Anwendungsfalldiagrammen erstellen (siehe Seite 220)

10.9.12 Überblick über LiveSource

LiveSource™ sorgt in Together dafür, dass Modell und Quelltext synchron bleiben. Dieses Feature ist nur für Implementierungsprojekte wirksam.

Wenn Sie ein Klassendiagramm in einem Implementierungsprojekt erstellen, wird es sofort mit dem Implementierungsquelltext synchronisiert. Bei Änderungen im Klassendiagramm aktualisiert Together automatisch den entsprechenden Quelltext.

Together bietet verschiedene Möglichkeiten, die Bestandteile eines Projekts zu synchronisieren.

Mit dem Befehl **Neu laden** können Sie veranlassen, dass Together das Modell auf Basis des Quelltextes aktualisiert.

MDA

Together unterstützt die modellgesteuerte Architektur (Model Driven Architecture, MDA) der Object Management Group (OMG).

MDA ist ein Architekturkonzept für herstellerneutrale Technologiespezifikationen, die eine modellgesteuerte Softwareentwicklung ermöglichen.

MDA wird von UML, XMI und von anderen Technologien unterstützt.

Dok-Kommentareigenschaften

Bestimmte Eigenschaften, die Sie im Objektinspektor für Modellelemente und Member festlegen, werden im Quelltext als sprachspezifische Dok-Kommentare dargestellt. Dies gilt für Eigenschaften wie *author*, *since*, *version*, *stereotype*, *associates* usw. Wenn solche Kommentare im Quelltext vorhanden sind, werden sie zu Modelleigenschaften rückentwickelt.

Dok-Kommentare sind XML-Tags mit vorangestelltem `///` (C#-Projekte).

Wenn die Eigenschaften eines Elements im alten Format angegeben sind und eine dieser Eigenschaften in das neue Format `<property> value</property>` konvertiert wird, werden auch alle anderen Eigenschaften umgewandelt.

Siehe auch

Modellansicht ([siehe Seite 262](#))

LiveSource-Regeln ([siehe Seite 1297](#))

Neu laden (Befehl in Modellansicht) ([siehe Seite 1270](#))

10.9.13 Überblick zur Umwandlung in Quelltext

Together ermöglicht es, Quelltext auf der Grundlage eines sprachneutralen Designprojekts zu generieren.

Quelltext aus Diagrammen erzeugen

Sie können aus den Klassendiagrammen eines Designprojekts Quelltext erzeugen und diesen zu einem anderen Projekt in einer der unterstützten Sprachen hinzufügen. Das Implementierungsprojekt, das als Ziel fungiert, muss bereits in derselben Projektgruppe vorhanden sein.

Es besteht auch die Möglichkeit, Quelltext aus einem externen Designprojekt in das aktuelle Implementierungsprojekt zu importieren.

Namenszuordnung

Sie können veranlassen, dass Together für Modellelemente im Quelltext andere Namen erzeugt. Beispielsweise könnte aus `Class1` im Modell `ClassItem` im Quelltext werden.

Diese Feature ist hilfreich, wenn die Modellnamen in einer bestimmten Sprache vorliegen und geändert werden sollen. So könnten etwa japanische Namen in Diagrammen im Quelltext in eine Sprache mit lateinischer Wurzel umgesetzt werden.

Wenn das Feature aktiviert ist, wird im Ordner für die Modellunterstützung des Quell-Designprojekts die Datei `codegen_map.xml` erstellt. Sie können diese Datei in einem XML- oder Texteditor öffnen und bearbeiten. Sie enthält eine Zuordnungstabelle, in der jedem Eintrag (Modellelement) zwei Namen zugewiesen sind: einer für das Quell-Designprojekt (Attribut `name`) und einer für das Ziel-Implementierungsprojekt (Attribut `alias`). Die Datei besteht aus folgenden Abschnitten: **Class**, **Attribute**, **Operation** und **Package** (für UML 1.5-Projekte) bzw. **Class** und **Package** (für UML 2.0-Projekte). Attributnamen für die Einträge innerhalb eines Abschnitts müssen eindeutig sein.

Sie können wahlweise eine XML-Datei mit demselben Namen und derselben Struktur im Ordner eines Pakets erstellen.

Wenn Sie das Projekt anschließend in Quelltext umwandeln, wird bei aktiver Zuordnungsfunktion für jedes Element nach der Datei `codegen_map.xml` gesucht. Ist die Datei für ein aktuelles Paket nicht vorhanden, sucht Together in einem übergeordneten Paket usw.

Anmerkung: Wenn Sie dem Modell später ein neues Element hinzufügen und das Projekt danach wieder in Quelltext umwandeln, fügt Together für das Element einen neuen Eintrag in die entsprechende `codegen_map.xml`-Datei ein. Vorhandene Einträge werden nicht verändert.

Siehe auch

LiveSource – Überblick (siehe Seite 1451)

Designprojekte in Quelltext umwandeln (siehe Seite 264)

10.9.14 Überblick zur OCL-Unterstützung

Was ist OCL?

OCL (Object Constraint Language) ist eine textbasierte Sprache, die speziell für den Einsatz mit diagrammbezogenen Sprachen wie UML entwickelt wurde. UML wurde um OCL erweitert, nachdem man erkannt hatte, dass die Ausdrucksmöglichkeiten in einer visuellen, diagrammbasierten Sprache begrenzt sind.

OCL 2.0 ist die neueste Version dieser Sprache, die für die objektorientierten Modellierungssprachen der Object Management Group (OMG) entwickelt wurde.

OCL ist als begeleitende Einschränkungs- und Abfragesprache für die Modellierung mit diesen Sprachen von größter Bedeutung.

Anmerkung: Komponenten dieses Produkts enthalten die Object Constraint Language Library der Kent University, Großbritannien. Siehe <http://www.cs.kent.ac.uk/projects/ocl/>

OCL-Einschränkungen und -Ausdrücke

OCL-Einschränkungen

Die Tool-Palette für bestimmte Diagrammtypen (z.B. UML 2.0-Klassendiagramme) enthält Schaltflächen, mit denen Sie **OCL-Einschränkungen** als Designelemente in Diagrammen erstellen und dann Beziehungen mit dem gewünschten **Kontext** herstellen können.

Sie können die Einschränkungselemente in Ihren Diagrammen je nach Bedarf ein- und ausblenden.

Die OCL-Unterstützung für Einschränkungen bietet eine Fehlerkennzeichnung. Der Text der Einschränkung wird automatisch überprüft, wenn Sie die Beziehung mit dem Kontext herstellen. Die gültigen Einschränkungen werden in normaler Schrift angezeigt. Ungültige Einschränkungen und OCL-Ausdrücke mit Syntaxfehlern werden in roter Schrift angezeigt.

Die Kontextelemente der Einschränkungen werden durch kleine Symbole gekennzeichnet. Bei einer gültigen Einschränkung ist das Symbol grün, andernfalls rot. Wenn die Einschränkungen ausgeblendet sind, kann ihre Gültigkeit weiterhin über die Symbole überwacht werden.

Jede OCL-Einschränkung enthält einen **OCL-Ausdruck**.

OCL-Ausdrücke

Bei OCL-Ausdrücken ohne Objekteinschränkungen (Ausdrücke als Eigenschaften anderer Knoten) findet keine Überprüfung statt, da bei diesen Elementen kein gültiger OCL-Kontext festgelegt werden kann.

Unterstützte Diagrammtypen

OCL wird für die folgenden Diagrammtypen unterstützt:

Diagrammtypen mit OCL-Unterstützung

Diagrammtyp	UML-Version	Bereitstellung der Unterstützung
Klasse (Klasse, Namespace, Paket)	UML 1.5, 2.0	Es können Objekteinschränkungen erstellt werden.
Interaktion (Sequenz und Kommunikation)	UML 2.0	Zustandsinvariantenbeschränkungen für Lebenslinien und Einschränkungen für die Operanden kombinierter Fragmente als OCL-Ausdrücke.
Zustandsmaschine	UML 2.0	Überwachungsbedingungen (Guards) von Übergängen als OCL-Ausdrücke.
Anwendungsfall	UML 2.0	Vor- und Nachbedingungseinschränkungen für das einem Anwendungsfall zugeordnete Verhalten als OCL-Ausdrücke, z.B. eine Interaktion.

Siehe auch

Überblick zur Modellierung (siehe Seite 1446)

Together Object Constraint Language (OCL) – Anleitungen (siehe Seite 1397)

OCL-Editor (Diagrammansicht) (siehe Seite 1269)

10.9.15 Überblick über Pattern

Mit **Pattern** stehen dem Software-Entwickler leistungsstarke Wiederverwendungstools zur Verfügung. Sie müssen dann nicht jedes Entwurfsproblem von vorne angehen, sondern können die vordefinierten Pattern von Together verwenden. Die Hierarchie der Pattern wird in der Pattern-Registrierung festgelegt. Mit Hilfe des Pattern-Organizer können Sie Pattern verwalten und logisch anordnen.

Pattern werden für folgende Zwecke eingesetzt:

- Erstellung häufig verwendeter Elemente
- Änderung vorhandener Elemente
- Implementierung von nützlichen Quelltextkonstrukten oder Projektgruppen in Modellen

Pattern-Registrierung

Die Pattern-Registrierung definiert die virtuelle Hierarchie der Pattern. Sie können gemäß den jeweiligen Projektanforderungen virtuelle Ordner erstellen und die Pattern nach logischen Gesichtspunkten gruppieren. Alle Operationen mit dem Inhalt der Pattern-Registrierung werden im Pattern-Organizer durchgeführt und mit der Pattern-Registrierung synchronisiert.

Pattern-Organizer

Mit Hilfe des Pattern-Organizers können Sie Pattern logisch organisieren (mit virtuellen Bäumen, Ordnern und Verknüpfungen) und die Eigenschaften von Pattern anzeigen und bearbeiten. Dabei arbeiten Sie nicht mit den Pattern selbst, sondern mit ihren Verknüpfungen. Daher können Verknüpfungen mit demselben Pattern in mehreren Ordnern vorhanden sein.

Quelltext-Templates

Quelltext-Templates sorgen in Together für die Abwärtskompatibilität zu vorhandenen Together ControlCenter-Projekten. Sie können die Ordner mit den vorhandenen Quelltext-Templates in den Unterordner Patterns des Together-Installationsverzeichnisses kopieren und danach mit Hilfe dieser Templates Elemente in den Implementierungsprojekten erzeugen.

Quelltext-Templates sind Textdateien mit sprachspezifischen Namenserweiterungen. Sie enthalten Makros, die beim Anwenden der Templates durch die tatsächlichen Werte ersetzt werden. Aus diesem Grund lassen sich Quelltext-Templates als Vorlagen für bestimmte Anwendungszwecke nutzen, die jeweils nur entsprechend "ausgefüllt" werden müssen. Eine Quelltext-Template besteht aus einer Template-Datei mit Quelltext und einer Eigenschaftsdatei mit Makrobeschreibungen und den dazugehörigen

Standardwerten.

Quelltext-Templates werden im Verzeichnis `Patterns\templates` der Together-Installation mit folgender Struktur gespeichert:

`/<Sprache>/<Kategorie>/<Template-Name>`

Dabei steht `<Kategorie>` für CLASS, LINK oder MEMBER. Jeder `<Template_Name>`-Ordner enthält die folgenden Dateien:

- `%Name%.<ext>`
- `<template_name>.properties` (optional)

Entwurfspattern

Ein Entwurfspattern ist eine XML-Datei, die Anweisungen oder Aktionen zum Erstellen von Entitäten und Beziehungen sowie zum Festlegen ihrer Eigenschaften enthält. Mit jeder Anweisung wird entweder ein Modellelement oder eine Beziehung zwischen Modellelementen erstellt.

Sie können mit Entwurfspattern nicht nur neue Elemente erzeugen, sondern auch einem Container-Element Member hinzufügen. Die Eigenschaft `Use Exist` des Pattern, das auf ein bestimmtes Container-Element angewendet wird, muss auf `True` gesetzt sein. Sie können dann das Pattern auf das Container-Element anwenden, das geändert werden soll. Wenn Sie beispielsweise einer vorhandenen Klasse mehrere Methoden hinzufügen möchten, die in einer anderen Klasse als Pattern gespeichert sind, müssen Sie dieses Pattern auf das Diagramm anwenden, in dem sich die Klasse befindet.

Die Entwurfspattern sind als XML-Dateien im Verzeichnis `Patterns` der Together-Installation gespeichert.

Pattern als First-Class-Citizens

Ein First-Class-Citizen (FCC)-Pattern ist ein spezielles Entwurfspattern, das Informationen über den Pattern-Namen und die Rolle jedes Teilnehmers enthält. Wenn FCC-Pattern auf ein Diagramm angewendet werden, erzeugen sie eigene Entitäten und werden im Diagramm mit Beziehungslinien zu den erstellten Entitäten angezeigt. Diese Pattern ermöglichen weitere Änderungen durch Hinzufügen neuer Teilnehmer.

Pattern als First-Class-Citizens werden durch GoF-Pattern repräsentiert.

Ein Pattern wird in einem Diagramm als Oval mit dem Pattern-Namen und einer erweiterbaren Liste der Teilnehmer angezeigt. Jeder Teilnehmer ist mit dem Pattern-Oval durch eine Beziehung verbunden, die mit der Rolle des Teilnehmers beschriftet ist.

FCC-Pattern erzeugen Quelltext, die ovalen FCC-Pattern-Elemente jedoch nicht. Die von Pattern erzeugten Entitäten werden in den Diagrammdateien gespeichert.

Stub-Implementierungs-Pattern

Wenn Sie eine Vererbungsbeziehung zwischen einer Klasse und einer anderen abstrakten Klasse oder einem Interface erstellen, werden die Methoden und Member nicht automatisch den untergeordneten Klassen hinzugefügt. Dieses Problem lässt sich mit dem Stub-Implementierungs-Pattern lösen. Mit Hilfe des Pattern `Implementation link and stub` können Sie eine Implementierungsbeziehung und eine Stub-Implementierung in einem einzigen Arbeitsschritt erzeugen.

Ist das Ziel einer Beziehung ein Interface, veranlasst das Pattern, dass die Ausgangsklasse das Interface implementiert, und erstellt in einer Klasse die Stubs für alle Methoden des Interface und aller seiner übergeordneten Interfaces.

Wenn das Ziel einer Beziehung eine abstrakte Klasse ist, veranlasst das Pattern eine Erweiterung der Zielklasse durch die Ausgangsklasse und erzeugt Stubs für alle Konstruktoren der Zielklasse. Diese Konstruktor-Stubs rufen die entsprechenden Konstruktoren der Zielklasse auf.

Das Pattern `Implementation link and stub` wird über den Pattern-Experten bereitgestellt. Klicken Sie in der Tool-Palette auf die Schaltfläche **Beziehung nach Pattern** oder **Knoten nach Pattern**. Sie können aber auch den Befehl **Nach Pattern erstellen** im Kontextmenü einer Klasse verwenden.

Das Pattern `Implementation link and stub` erzeugt die folgenden Member von Interfaces oder abstrakten Klassen:

- Methoden

- Funktionen
- Subroutinen
- Eigenschaften
- Indexierer
- Ereignisse

Siehe auch

Together-Pattern – Anleitungen ([siehe Seite 1398](#))

Pattern-Organizer ([siehe Seite 1271](#))

Pattern-Registrierung ([siehe Seite 1272](#))

10.9.16 Überblick zum Refactoring

Together bietet eine umfassende Unterstützung für das Refactoring von Implementierungsprojekten.

Unter Refactoring versteht man die Umgestaltung vorhandenen Quelltextes mit dem Ziel, seine Struktur zu optimieren. Auf das Verhalten der Anwendung hat das Refactoring keinerlei Auswirkungen. Nach dem Refactoring lässt sich der Quelltext einfacher interpretieren, pflegen und bearbeiten.

Die Refactoring-Funktionen von Together wirken sich sowohl auf den Quelltext als auch auf das Modell aus. Nach dem Refactoring ist das Projekt deshalb immer konsistent, auch dann, wenn es UML-Diagramme enthält.

Basiswissen zum Refactoring finden Sie im Buch *Refactoring. Wie Sie das Design vorhandener Software verbessern* von Martin Fowler (Addison-Wesley Verlag, 1999).

Siehe auch

Refactoring-Operationen anwenden ([siehe Seite 1401](#))

Refactoring-Operationen – Referenz ([siehe Seite 1331](#))

10.9.17 Überblick über Qualitätssicherungsfunktionen

Together stellt Audits und Metriken als Qualitätssicherungsfunktionen bereit, mit deren Hilfe unternehmensweite Programmierstandards und -konventionen eingehalten, echte Metriken erfasst und der Programmierstil verbessert werden kann. Obwohl sich Audits und Metriken ähneln – beide Operationen analysieren Quelltext – dienen sie doch unterschiedlichen Zwecken.

Audits und Metriken werden als separate Prozesse ausgeführt. Da die Ergebnisse beider Prozesse unterschiedlicher Natur sind, stellt Together verschiedene Funktionen für die Interpretation und Organisation der Ergebnisse bereit. Einige der in diesem Abschnitt beschriebenen Funktionen und Verfahren gelten sowohl für Audits als auch für Metriken, während andere nur einen der beiden Prozesse betreffen.

Audits

Beim Ausführen von Audits werden bestimmte Regeln ausgewählt, mit denen der Quelltext übereinstimmen muss. Anschließend werden im Ergebnis die Verletzungen dieser Regeln angezeigt, sodass Sie die Probleme einzeln untersuchen und entscheiden können, ob der Quelltext entsprechend korrigiert werden muss. Together bietet eine Vielzahl verschiedener Audits, die von Entwurfsproblemen bis zu Namenskonventionen reichen. Außerdem werden Beschreibungen der Funktionsweise der Audits und

Vorschläge zum Korrigieren der Regelverletzungen angezeigt. Sie haben die Möglichkeit, Audit-Sets zu erstellen und diese zur Ausführung bzw. Wiederverwendung zu speichern. Together wird mit einem vordefinierten Audit-Set (`current.adt`) ausgeliefert. Sie können aber auch eigene Audit-Sets erzeugen.

Warnung: Diese Funktion steht nur für Implementierungsprojekte zur Verfügung.

Metriken

Metrikanalysen bewerten die Komplexität des Objektmodells und quantifizieren Ihren Quelltext. Es liegt dann an Ihnen, das Ergebnis zu überprüfen und zu entscheiden, ob es akzeptabel ist. Anhand des Ergebnisses der Metrik können Sie erkennen, welche Teile des Quelltextes geändert werden müssen. Sie können auch Berichte erstellen, um die Gesamtauswirkungen der Änderungen in einem Projekt zu vergleichen.

Together unterstützt eine breite Palette von Metriken. Weitere Informationen finden Sie bei der Beschreibung der verfügbaren Metriken im Dialogfeld **Metriken**.

Sie können Metrik-Sets definieren und zur Ausführung bzw. Wiederverwendung speichern.

Zu allen Metriken stellt Together auch Tipps für ihre Verwendung und für die Auswertung der Ergebnisse zur Verfügung.

Warnung: Diese Funktion steht nur für Implementierungsprojekte zur Verfügung.

Balkendiagramm

Metrikergebnisse können auch grafisch angezeigt werden. Es gibt zwei grafische Ansichten, in denen Sie die Metrik-Ergebnisse zusammenfassen können: Balken- und Kiviat-Diagramme. Beide Diagramme werden im Kontextmenü der Ergebnistabelle ausgewählt. Verwenden Sie das Kiviat-Diagramm für Zeilen und das Balkendiagramm für Spalten.

Das Balkendiagramm zeigt die Ergebnisse einer bestimmten Metrik für alle Pakete, Klassen und/oder Methoden an.

Die Farbe des Balkens gibt an, wie sich die betreffende Metrik zum festgelegten Grenzwert verhält:

- Ein grüner Balken steht für Werte, die sich im erlaubten Bereich bewegen.
- Ein roter Balken steht für Werte, die den oberen Grenzwert überschreiten.
- Ein blauer Balken steht für Werte, die den unteren Grenzwert unterschreiten.
- Eine dünne senkrechte rote Linie markiert den oberen, eine dünne blaue vertikale Linie den unteren Grenzwert.

Kiviat-Diagramm

Verwenden Sie das Kiviat-Diagramm für Zeilen und das Balkendiagramm für Spalten.

Das Kiviat-Diagramm zeigt das Ergebnis der Analyse für die aktuell ausgewählte Klasse oder das aktuell ausgewählte Paket an, berücksichtigt dabei aber nur Metriken mit vordefinierten Grenzwerten. Die Ergebnisse der einzelnen Metriken werden auf Achsen abgebildet, deren Ursprung im Mittelpunkt des Graphen liegt.

Die Achsen sind logarithmisch, wobei die logarithmische Basis den oberen Grenzwert der Achse darstellt und daher alle oberen Grenzwerte dieselbe Entfernung vom Mittelpunkt haben. Die Werte und Grenzwerte werden dabei folgendermaßen angezeigt:

- Ein roter Kreis stellt die oberen Grenzwerte dar. Alle Punkte außerhalb des Kreises verletzen den oberen Grenzwert.
- Eine blaue Schattierung zeigt die unteren Grenzwerte an. Alle Punkte innerhalb der Schattierung verletzen den unteren Grenzwert. Bereiche mit einer Untergrenze von 1 oder 0 sind nicht schattiert.

Wenn sich der Mauszeiger auf dem Diagramm befindet, werden in der Statusleiste von Visual Studio Informationen über die Metriken bzw. die Metrikwerte angezeigt, die den Markierungen entsprechen.

- Alle Metriken bilden einen Stern. Die Werte der einzelnen Metriken sind als Punkte dargestellt.
- Grüne Punkte stehen für akzeptable Werte.
- Blaue Punkte stehen für Werte, die den unteren Grenzwert unterschreiten.
- Rote Punkte stehen für Werte, die den oberen Grenzwert überschreiten.
- Im rechten Winkel zu den einzelnen Strahlen des Kiviat-Graphen sind im Uhrzeigersinn Skalierungsmarken angeordnet.
- Die blauen Marken gegen den Uhrzeigersinn stellen die Untergrenzen dar.

Audit- und Metrik-Sets

In den Dialogfeldern **Audits** und **Metriken** werden die verfügbaren Audit- bzw. Metrik-Sets angezeigt. Wenn Sie ein Projekt öffnen, ist automatisch ein Standard-Set aktiviert. Die aktivierten Audits und Metriken sind mit einem Häkchen gekennzeichnet. Wenn Sie das Dialogfeld öffnen und auf **Start** klicken, werden alle aktivierten Audits/Metriken ausgeführt.

In der Regel werden Sie aber nicht jedes Mal sämtliche Audits oder Metriken im Standard-Set ausführen wollen, sondern nur eine bestimmte Teilmenge. Together ermöglicht es Ihnen, eigene Sets mit Audits und Metriken zu speichern, um sie bei Bedarf neu zu laden und auszuführen. Zu diesem Zweck stehen in den Dialogfeldern **Audits** und **Metriken** die Schaltflächen **Audit-Set laden** und **Audit-Set speichern** bzw. **Metrik-Set laden** und **Metrik-Set speichern** zur Verfügung. Sie können das Standard-Set jederzeit über die Schaltfläche **Standard-Audit-Set festlegen** im Dialogfeld **Audits** wiederherstellen. Eine Beschreibung der Steuerelemente finden Sie im Thema für das Dialogfeld **Audits**.

Verwenden Sie das Standard-Set oder ein gespeichertes Set als Ausgangspunkt für die Erstellung neuer Sets. Audit-Sets werden standardmäßig im QS-Ordner der Together-Installation gespeichert.

Siehe auch

Audits ausführen (siehe Seite 269)

Metriken ausführen (siehe Seite 273)

10.9.18 Überblick zur Dokumentationserzeugung

Sie können für Ihre Projekte automatisch Dokumentation generieren lassen. Mit der Funktion zur automatischen Dokumentationserzeugung erhalten Sie Dokumentation im HTML-Format. Die erzeugte Dokumentation kann bei Änderungen im Projekt automatisch aktualisiert werden. Sie können die Änderungen aber auch nachträglich manuell in die Dokumentation einfügen.

Dokumentationsdateien

Alle von Together erzeugten Dokumentationen werden in dem Verzeichnis abgelegt, das Sie im Dialogfeld Dokumentation erzeugen als Ausgabeordner festlegen. Erzeugte Dokumentation wird standardmäßig in den externen Web-Browser geladen. Die Anzeige erfolgt in einem Browser-Frameset. Wenn Sie die Dokumentation nicht sofort laden möchten, können Sie dies später nachholen, indem Sie die Datei `index.html` öffnen. Diese befindet sich im Stammverzeichnis des Dokumentationsordners, den Sie im Dialogfeld Dokumentation erzeugen angegeben haben.

Frames für HTML-Dokumentation

Die HTML-Dokumentation wird in drei Frames angezeigt:

- Diagramm-Frame (wenn die Option Diagramme einbeziehen aktiviert ist).
- Projekt- und Übersichts-Frame (wenn die Option Navigationshierarchie einbeziehen aktiviert ist)
- Paketlisten- und Paketübersichts-Frame (wenn die Option Navigationshierarchie einbeziehen aktiviert ist)
- Dokumentations-Frame

Durch Klicken auf das Register Projekt im unteren linken Frame können Sie die Knoten in der Projekt-Strukturansicht erweitern.

Wenn Sie auf der Registerkarte Projekt auf einen Klassennamen klicken, wird die Dokumentation im unteren rechten Frame (dem Dokumentations-Frame) angezeigt. Wenn Sie auf der Registerkarte Projekt ein Diagramm auswählen, wird es im Diagramm-Frame geöffnet. Die Elemente im Diagramm-Frame sind per Hyperlink mit dem Dokumentations-Frame verknüpft. Wenn Sie ein Element im Diagramm-Frame auswählen, wird sein Inhalt im Dokumentations-Frame angezeigt.

Im Dokumentations-Frame wird die Dokumentation für Ihren Quelltext und Ihre Diagramme angezeigt. Der Frame enthält alle Komponenten, die Sie von einer HTML-Dokumentation erwarten. Am oberen Rand des Dokumentations-Frame befindet sich eine Navigationsleiste, mit der Sie die Projektdokumentation durchgehen können.

Die Registerkarte Projekt enthält eine baumartige Darstellung des Projekts. Blenden Sie die Knoten ein, um die verschiedenen Diagramme und Elemente anzuzeigen. Durch Klicken auf eine Klasse oder ein Interface öffnen Sie die entsprechende Dokumentation im Dokumentations-Frame.

Siehe auch

Projektdokumentation erzeugen (siehe Seite 224)

Funktion zur Dokumentationserzeugung konfigurieren (siehe Seite 223)

10.9.19 Überblick zum Importieren und Exportieren

Sie können Modellinformationen systemübergreifend nutzen, indem Sie sie importieren und exportieren oder Projektdateien für die gemeinsame Nutzung bereitstellen.

Funktionen für den Import und Export

Funktion	Beschreibung
Exportieren von Diagrammen als Grafik	Diagramme können in folgenden Formaten gespeichert werden: Bitmap-Grafik (BMP) Enhanced Windows Metafile (EMF) Graphics Interchange (GIF) JPEG File Interchange (JPG) W3C Portable Network Graphics (PNG) Tag Image File (TIFF) Windows Metafile (WMF)
Importieren von IBM Rational Rose-Modellen (MDL)	Modelle, die mit IBM Rational Rose 2003 entwickelt wurden, können in das Together-Format konvertiert werden. Folgende Dateiformate werden unterstützt: .mdl, .ptl, .cat und .sub. Für den Import wird ein neues UML 1.5-Designprojekt erstellt, das auf dem IBM Rational Rose-Projekt basiert.
Importieren aus XMI Exportieren nach XMI	XMI (XML Metadata Interchange) ist ein Format zum Austausch von Metadaten. Mit seiner Hilfe können Modelle zwischen Programmiersprachen und Anwendungen ausgetauscht werden. Beispielsweise können Sie ein Modellprojekt, das mit einem externen Tool erstellt wurde, in Form einer XMI-Datei in Together importieren und als Erweiterung für ein vorhandenes oder als Grundlage für ein neues Projekt verwenden. Ebenso lassen sich Together-Projekte in andere Anwendungen exportieren. Das Ergebnis ist in beiden Fällen eine einzelne, portierbare .xml-Datei. Together unterstützt den UML 1.3-Unisys-XMI-Austausch für acht UML-Diagrammtypen. Diese Funktion ist für Designprojekte verfügbar, die mit der UML 1.5-Spezifikation konform sind.
Importieren aus anderen Together-Versionen	Siehe Überblick zur Interoperabilität (siehe Seite 1460)

Exportieren eines QS-Metrikdiagramms in eine Grafik	Sie können ein Diagramm nach der Erstellung in eine Grafik exportieren.
---	---

Siehe auch

Diagramm als Grafik exportieren ([siehe Seite 136](#))

Projekt im IBM Rational Rose-Format (MDL) importieren ([siehe Seite 251](#))

Projekt im XMI-Format importieren ([siehe Seite 256](#))

Projekt in das XMI-Format exportieren ([siehe Seite 250](#))

Diagramme erstellen ([siehe Seite 272](#))

Unterstützte Projektformate ([siehe Seite 1332](#))

10.9.20 Überblick zur Interoperabilität

Diese Version von Together ist mit anderen Versionen kompatibel. Aufgrund eines gemeinsamen Diagrammformats können Modelle wieder verwendet werden, die in anderen Editionen von Together erstellt wurden:

- CodeGear Together ControlCenter (TCC)
- CodeGear Together Architect (TAR)
- CodeGear Together für Microsoft Visual Studio .NET (TVS)
- CodeGear Together für Eclipse (TEC)
- CodeGear Together für JBuilder (TJB)
- CodeGear Together Designer 2005 und CodeGear Together Developer 2005 für PrimeTime (TPT)

Siehe auch

In TCC oder TAR erstellte Projekte importieren ([siehe Seite 252](#))

In TVS ([siehe Seite 254](#))

Gemeinsame Nutzung eines Projekts in TCC/TAR und RAD Studio ([siehe Seite 258](#))

Allgemeine Optionen von Together ([siehe Seite 1281](#))

11 Überblick zu MSBuild

Die IDE verwendet nun zum Erstellen von Projekten MSBuild anstelle des früheren internen Build-Systems. Die Befehle zum Compilieren und Erzeugen, die in der IDE zur Verfügung stehen, rufen die neue Build-Engine von Microsoft auf: MSBuild, das gründliche Abhängigkeitsprüfung bietet. MSBuild-Projektdateien basieren auf XML und enthalten Abschnitte, die spezielle Einträge, Eigenschaften, Aufgaben und Ziele für das Projekt beschreiben.

Weitere Informationen über MSBuild finden Sie in der Microsoft-Dokumentation unter <http://msdn.microsoft.com>.

Projekte nach MSBuild migrieren

Wenn Sie ein bereits vorhandenes Delphi-Projekt (wie z.B. eines mit der Erweiterung `.bdsproj`) öffnen, konvertiert die IDE das Projekt automatisch für die Verwendung von MSBuild und ändert die Projekterweiterung für ein Delphi-Projekt in `.dproj` oder in `.cbproj` für ein C++-Projekt.

Projektgruppen werden auch in MSBuild konvertiert und erhalten die Projektgruppenerweiterung `.groupproj`.

Projekte erzeugen

Sie können ohne Kenntnisse über MSBuild Projekte erstellen; weil die IDE alle Details für Sie handhabt. Die Befehle **Projekt**▸**Compilieren** und **Projekt**▸**Erzeugen** rufen jeweils MSBuild auf, aber der Befehlsumfang ist unterschiedlich.

Projekte lassen sich auch explizit von der Befehlszeile aus mittels MSBuild und Ihrer `.dproj`-Datei erstellen.

Zum Aufrufen von MSBuild in einer benutzerdefinierten Befehlsumgebung können Sie **Startmenü**▸**RAD Studio-Befehlszeile** wählen. Das Befehlsfenster setzt automatisch den Pfad zu der ausführbaren Datei und die Variable für Ihr Installationsverzeichnis.

Eigene Build-Konfigurationen anwenden

Auf verschiedenen Seiten des Dialogfeldes **Projekt**▸**Optionen** können Sie Optionsgruppen als benannte Build-Konfiguration speichern. Zwei Standard-Build-Konfigurationen sind **Debug** und **Ausgabe**. C++Builder unterstützt auch die Konfiguration **Base**. Mit dem Konfigurations-Manager kann jede benannte Build-Konfiguration selektiv einem Projekt oder einer Projektgruppe als aktive Build-Konfiguration zugewiesen werden.

Build-Ereignisse setzen und die Build-Ausgabe anzeigen

Wenn Sie ein Projekt-Build erstellen, erscheinen die Ergebnisse des Build-Prozesses auf der Registerkarte **Ausgabe** des Fensters **Meldungen**. Sie können im Dialogfeld **Projekt**▸**Optionen**▸**Build-Ereignisse** Pre-Build- und Post-Build-Ereignisse festlegen (C++Builder unterstützt auch Pre-Link-Ereignisse). Wenn Sie Build-Ereignisse festgelegt haben, erscheinen die Befehle und deren Ergebnisse im Fenster **Meldungen**. Verwenden Sie zum Steuern der Ausgabeebene von MSBuild das Feld

Verbosity auf der Registerkarte **Tools**▶**Optionen**▶**Umgebungsoptionen**.

Der Dateityp bestimmt die Build-Reihenfolge

MSBuild erzeugt ein Projekt anhand der folgenden Reihenfolge:

1. .RC-Dateien
2. .ASM-Dateien
3. .PAS-Dateien
4. .CPP-Dateien

Der Build-Prozess wird von oben nach unten durch die Verzeichnis- oder Ordnerknoten in der Projektverwaltung ausgeführt. In jedem Ordner werden die Dateien entsprechend ihres Dateityps erzeugt. Sie können die Build-Reihenfolge steuern, indem Sie Dateien in unterschiedliche Ordner oder in virtuelle Ordner in der Projektverwaltung platzieren.

Siehe auch

Compilieren (siehe Seite 1351)

Festlegen von Standardprojektoptionen (C++) (siehe Seite 76)

Build-Ereignisse (Dialogfeld) (siehe Seite 497)

Ausführen von MSBuild von der Befehlszeile aus (siehe Seite 12)

Überblick über Build-Konfigurationen (Delphi) (siehe Seite 1463)

Überblick über Build-Konfigurationen (C++) (siehe Seite 1465)

Erstellen von benannten Build-Konfigurationen (C++) (siehe Seite 6)

Übernehmen der aktiven Build-Konfiguration (siehe Seite 2)

Überblick über virtuelle Ordner

12 Überblick über Build-Konfigurationen (Delphi)

Build-Konfigurationen bestehen aus Optionen, die Sie auf allen mit dem Erzeugen zusammenhängenden Registerkarten des Dialogfeldes **Projekt>Optionen** festlegen können. Die Informationen der Build-Konfigurationen werden in der MSBuild-Projektdatei (`.dproj` oder `.groupproj`) gespeichert.

Die IDE enthält drei Standard-Build-Konfigurationen

Debug und **Ausgabe** sind die beiden Standard-Build-Konfigurationen. Die **Debug**-Konfiguration aktiviert z.B. die Optimierung und das Debugging und das Setzen bestimmter Syntaxoptionen.

Sie können auf den entsprechenden Registerkarten des Dialogfelds **Projekt>Optionen** die Optionen für die drei Standard-Konfigurationen ändern und eigene Build-Konfigurationen erstellen und benennen.

Sie können die aktive Konfiguration für Ihre Projekte festlegen

Allen Projekten in RAD Studio ist eine aktive Build-Konfiguration zugewiesen, sowie weitere inaktive Build-Konfigurationen, die Sie erstellt haben.

Die aktive Build-Konfiguration wird von den Befehlen Compilieren und Erzeugen für dieses Projekt verwendet. Legen Sie im Manager für Build-Konfigurationen die Konfiguration fest, die die aktive Konfiguration für ein ausgewähltes Projekt oder eine Projektgruppe sein soll (wählen Sie **Projekt>Konfigurations-Manager**).

Siehe auch

Überblick zu MSBuild (siehe Seite 1461)

Erstellen von benannten Build-Konfigurationen (siehe Seite 6)

Übernehmen der aktiven Build-Konfiguration (siehe Seite 2)

13 Überblick über Build-Konfigurationen (C++)

Build-Konfigurationen bestehen aus Optionen, die Sie auf allen mit dem Erzeugen zusammenhängenden Registerkarten des Dialogfeldes **Projekt**▸**Optionen** festlegen können. Build-Konfigurationen werden in der Projektdatei (`.cbproj` oder `.groupproj`) gespeichert.

Jedes Projekt enthält Standard-Build-Konfigurationen

Base, **Debug** und **Ausgabe** sind die drei Standard-Build-Konfigurationen. **Base** dient als Basisgruppe von Optionswerten, die in allen folgenden Konfigurationen, die Sie erstellen, verwendet wird. Die **Debug**-Konfiguration aktiviert die Optimierung und das Debugging und das Setzen bestimmter Syntaxoptionen.

Sie können in allen Konfigurationen, auch in **Base**, Optionswerte ändern. Sie können die Konfigurationen **Debug** und **Release** löschen, aber nicht die Konfiguration **Base**.

Sie können die aktive Konfiguration für Ihre Projekte festlegen

Allen Projekten ist eine aktive Build-Konfiguration zugewiesen, sowie weitere inaktive Build-Konfigurationen, die Sie erstellt haben.

Die aktive Build-Konfiguration wird von den Befehlen Compilieren und Erzeugen für dieses Projekt verwendet. Legen Sie im Manager für Build-Konfigurationen die Konfiguration fest, die die aktive Konfiguration für ein ausgewähltes Projekt oder eine Projektgruppe sein soll (wählen Sie **Projekt**▸**Konfigurations-Manager**).

Build-Konfigurationen

Jede Konfiguration, außer **Base**, basiert auf einer anderen Konfiguration, die ihre Werte vererbt. Die Konfigurationen **Debug** und **Ausgabe** erben ihre Werte von **Base**.

Sie können eine neue Konfiguration erstellen, die auf einer beliebigen anderen Konfiguration basiert. Die neue Konfiguration erbt die Werte von ihrem Vorgänger. Nach dem Erstellen einer Konfiguration lassen sich deren Werte beliebig ändern, und Sie können sie als aktive Konfiguration für ein oder mehrere Projekte übernehmen. Alle Konfigurationen, außer **Base**, können auch gelöscht werden.

Wenn eine Konfiguration nicht verändert wird, erbt sie ihre Optionswerte von der übergeordneten Konfiguration. Diese Vererbung ist nicht statisch: wenn die übergeordnete Konfiguration geändert wird, ändern sich auch alle geerbten Werte der untergeordneten Konfigurationen.

Der Standardwert einer Option ist ihr Wert in der übergeordneten Konfiguration. Sie können eine Option auf ihren Standardwert zurücksetzen.

Konfigurationen und Optionsgruppen im Vergleich

Sie können die Optionswerte einer Konfiguration auch mit Hilfe eines Speichern-Dialogfeldes als Datei mit einer *benannten Optionsgruppe* speichern. Optionsgruppenwerte lassen sich auf jede beliebige Konfiguration eines jeden Projekts anwenden.

Beachten Sie bitte, dass eine Konfiguration nicht dasselbe wie eine Optionsgruppe ist, obwohl sie Ähnlichkeiten aufweisen. Beide enthalten eine Gruppe von Optionswerten. Der Hauptunterschied besteht darin, dass Konfigurationen mit Projekten verbunden sind, während Optionsgruppen als von Projekten unabhängige Dateien gespeichert werden. Build-Konfigurationswerte werden in der Projektdatei gespeichert. Beim Speichern eines Projekts werden daher die Änderungen der Konfigurationen gespeichert, aber die Optionsgruppen bleiben unverändert. Das Ändern der Konfigurationen eines Projekts und das Hinzufügen oder Löschen von Konfigurationen wirkt sich nicht auf die Optionsgruppen aus. Genauso verändert das Speichern von Optionsgruppen die Konfigurationen nicht.

Die Drop-Down-Liste Build-Konfiguration enthält Konfigurationen für das jeweilige Projekt — keine Optionsgruppen. Jedes Projekt verfügt — unabhängig von anderen Projekten — über eine eigene Konfigurationsliste. Sie können jedoch auf alle Optionsgruppen aus allen Projekten zugreifen.

Konfigurations- und Optionsgruppenwerte

Beachten Sie bitte, dass Konfigurationen und Optionsgruppen nicht für alle möglichen Projektoptionen Werte enthalten müssen. Sie enthalten *nur* die Optionen, die sich von der übergeordneten Konfiguration unterscheiden. Auch **Base** enthält nicht für alle möglichen Optionen Werte.

Wenn ein Optionswert nicht in einer Konfiguration enthalten ist, sucht die IDE in der übergeordneten Konfiguration, dann in der übergeordneten Konfiguration übergeordneten Konfiguration usw. Wird der Wert in keiner Konfiguration der Vererbungskette gefunden, wird er von dem entsprechenden Tool, das konfiguriert wird, festgelegt.

Wenn eine Konfigurationsvererbung beispielsweise keinen Wert für eine bestimmte Compiler-Option enthält, wird der Vorgabewert vom Compiler selbst festgelegt. Wenn Sie eine Konfiguration oder eine Optionsgruppe speichern, werden nur diese Werte gespeichert, nicht Werte für alle Optionen.

Sie können Optionen aus der übergeordneten Konfiguration zusammenführen

Neben einigen Optionen, die eine Liste mit Einträgen enthalten, wie z.B. Definitionen oder Pfadangaben, wird das Kontrollfeld Zusammenführen angezeigt.

Wenn dieses Kontrollfeld aktiviert ist, führt die IDE die Optionsliste mit der des unmittelbaren Vorfahren der Konfigurationsliste für diese Option zusammen. Beachten Sie, dass die IDE den Inhalt der Option nicht eigentlich ändert, sondern sich so verhält, als ob die Liste die Liste des Vorfahren enthielte. Wenn das Kontrollfeld Zusammenführen des Vorfahren auch aktiviert ist, führt die IDE die Liste dieses Vorfahren für diese Option zusammen, und weiter die Vererbungskette hinauf.

Ist das Kontrollfeld deaktiviert, verwendet die IDE nur die Einträge aus der aktuellen Konfiguration.

Einige Optionen stehen nicht mehr zur Verfügung

Einige Optionen aus früheren Produktversionen stehen nicht mehr zur Verfügung. Manche Optionen davon können jedoch noch über Options-Flags des jeweiligen Tools verwendet werden. Weitere Informationen finden Sie unter [Nicht verfügbare Optionen](#) (siehe Seite 587).

Siehe auch

Überblick zu MSBuild (siehe Seite 1461)

Erstellen von benannten Build-Konfigurationen (C++) (siehe Seite 6)

Übernehmen der aktiven Build-Konfiguration (siehe Seite 2)

Arbeiten mit benannten Optionsgruppen (siehe Seite 15)

Festlegen von C++-Standardprojektoptionen (siehe Seite 76)

Nicht verfügbare Optionen (siehe Seite 587)

14 Überblick über benannte Optionsgruppen

Benannte Optionsgruppen bestehen aus Optionen, die Sie auf den Build-bezogenen Seiten des Dialogfeldes [Projekt > Optionen](#) erstellen und dafür übernehmen können. Benannte Optionsgruppen werden in Dateien mit der Erweiterung `.optset` gespeichert.

Optionsgruppen erstellen

Sie können die Optionswerte eines Projekts über das Kontextmenü der Projektverwaltung und über die Schaltfläche Speichern der Dialogfelder der Projektoptionen als Datei mit einer *benannten Optionsgruppe* speichern.

Optionsgruppen und Konfigurationen im Vergleich

Optionsgruppen unterscheiden sich von Konfigurationen, obwohl sie Ähnlichkeiten aufweisen. Beide enthalten eine Gruppe von Optionswerten. Der Hauptunterschied besteht darin, dass Konfigurationen Bestandteil der Projektdatei sind, während Optionsgruppen als von Projekten unabhängige Dateien gespeichert werden. Beim Speichern eines Projekts werden Änderungen der Konfigurationen gespeichert; Optionsgruppen bleiben aber unverändert. Das Ändern der Konfigurationen eines Projekts und das Hinzufügen oder Löschen von Konfigurationen wirkt sich nicht auf die Optionsgruppen aus. Genauso verändert das Speichern von Optionsgruppen die Konfigurationen nicht. Jedes Projekt verfügt — unabhängig von anderen Projekten — über eine eigene Konfigurationsliste, nicht über eigene Optionsgruppen. Sie können auf alle Optionsgruppen aus allen Projekten zugreifen.

Konfigurations- und Optionsgruppenwerte

Konfigurationen und Optionsgruppen müssen nicht für alle möglichen Projektoptionen Werte enthalten. Wenn Sie eine Konfiguration oder eine Optionsgruppe speichern, werden nur die Werte gespeichert, die sich von den Werten der übergeordneten Konfiguration unterscheiden, nicht die Werte für alle Optionen. Eine Optionsgruppendatei verweist nicht auf eine Konfiguration eines Projekts, weil Optionsgruppen unabhängig von Konfigurationen sind.

Angenommen, Sie haben ein Projekt in der IDE geöffnet und die aktive Konfiguration ist Base. Sie ändern eine Option und speichern die Optionsgruppe. Die einzigen Optionswerte, die in der Optionsgruppendatei gespeichert werden, sind der eine Optionswert, den Sie geändert haben, plus der Optionswerte der Konfiguration Base, die von den Standardwerten abweichen.

Optionsgruppen anwenden

Sie können Optionsgruppenwerte mit Hilfe des Dialogfelds Optionsgruppe anwenden auf jede Konfiguration in jedem Projekt anwenden. Es gibt drei Möglichkeiten, die Werte zu übernehmen: Überschreiben, Ersetzen und Beibehalten.

- Überschreiben ersetzt die aktuelle Konfiguration vollständig durch die Werte aus der Optionsgruppe; Werte, die in der Optionsgruppe nicht enthalten sind, werden auf ihre Standardvorgaben gesetzt.
- Ersetzen schreibt alle Werte aus der Optionsgruppe in die aktuelle Konfiguration, ändert aber keine anderen Werte.
- Beibehalten schreibt nur die Werte aus der Optionsgruppe, die nicht schon in der aktiven Konfiguration gesetzt sind.

Siehe auch

Arbeiten mit benannten Optionsgruppen ([siehe Seite 15](#))

Optionsgruppe anwenden (Dialogfeld) ([siehe Seite 492](#))

Festlegen von C++-Standardprojektoptionen ([siehe Seite 76](#))

Erstellen von benannten Build-Konfigurationen ([siehe Seite 6](#))

Übernehmen der aktiven Build-Konfiguration ([siehe Seite 2](#))

15 Targets-Dateien

Eine targets-Datei ist eine MSBuild-konforme XML-Datei, die Sie Ihrem Projekt hinzufügen können, damit der Build-Prozess angepasst werden kann. Eine .targets-Datei kann **<Target>**-Knoten mit MSBuild-Scripts enthalten.

Mit einer .targets-Datei können auch Werte von Projekteigenschaften hinzugefügt und geändert werden. Sie können die vielfältigen MSBuild-Tasks, die .NET Framework SDK und das Internet bereitstellen, wie z.B. "Zip", "SVNCheckout" und "Mail" nutzen, oder selber benutzerdefinierte Tasks schreiben.

Zusammengefasst:

- Eine .targets-Datei ist eine XML-Datei mit **<Target>**-Knoten, die MSBuild-Scripts mit Listen von auszuführenden Tasks enthalten.
- Sie können .targets-Dateien in einem Projekt mit der IDE erstellen, hinzufügen, aktivieren oder entfernen.

.targets-Dateien hinzufügen

Sie erstellen und fügen eine .targets-Datei dem Projekt entweder mit Menübefehlen oder dem Kontextmenü der Projektverwaltung hinzu. Die IDE erzeugt eine minimale .targets-Datei, die nur den Stammknoten **<Project>** und das Attribut **namespace** enthält. Sie können dann in den Knoten **<Project>** MSBuild-Scripts einfügen.

Per Vorgabe werden dem Projekt .targets-Dateien hinzugefügt, aber nicht verwendet. Sie aktivieren eine .targets-Datei in der Projektverwaltung, die die .targets-Datei als einen MSBuild-<Import> zu Ihrem Projekt hinzufügt. Alle .targets-Dateien müssen gültige, fehlerfreie MSBuild-Scripts enthalten. Enthält eine Datei Fehler, werden Sie benachrichtigt. Wenn das Projekt die ungültige .targets-Datei referenziert, wird sie deaktiviert und kann erst wieder aktiviert werden, wenn die Fehler behoben sind. Weil MSBuild **<Import>** nur direkt vom Datenträger lesen kann, muss die .targets-Datei vor einem Compilier- oder Erzeugen-Befehl oder Aufruf eines Ziels gespeichert werden.

Target-Element in .targets-Dateien

Das Target-Element in der .targets-Datei enthält eine Gruppe von Tasks, die MSBuild ausführen soll. Es hat das folgende Format:

...

Weitere Informationen zu Target-Elementen finden Sie unter <http://msdn2.microsoft.com/en-us/library/t50z2hka.aspx>.

.targets-Dateien verwenden

Wenn eine .targets-Datei gültige **<Target>**-Elemente enthält, können Sie diese Ziele mit MSBuild aus der Projektverwaltung

aufrufen, vorausgesetzt, die .targets-Datei ist aktiviert.

Die .targets-Datei kann eigene Eigenschaften, Ziele und Elementgruppen für ihre Ziele und Tasks deklarieren. Sie kann auch auf Eigenschaften und Elementgruppen in der Projektdatei verweisen, auch auf solche, die aus der Standard-CodeGear-.targets-Datei im Windows-Verzeichnis unter **microsoft.net\Framework\v2.0.xxx\Borland.Cpp.Targets** importiert wurden. Sie sollten keine .targets-Dateien in diesem Verzeichnis ändern, weil die IDE durch falsche Bearbeitungen inkorrekt arbeiten könnte.

Tip: Weitere Informationen über .targets-Dateien finden Sie unter <http://msdn2.microsoft.com/en-us/library/ms164312.aspx>.

Siehe auch

Verwenden von Targets-Dateien (siehe Seite 14)

Index

"Cash Sales" ausführen 1384

\$

\$ALIGN

Compiler-Direktiven, Delphi 1052

\$APPTYPE

Compiler-Direktiven, Delphi 1053

\$ASSERTIONS

Compiler-Direktiven, Delphi 1053

\$AUTOBOX

Compiler-Direktiven, Delphi 1053

\$BOOLEVAL 1054

\$DEBUGINFO

Compiler-Direktiven, Delphi 1055

\$DEFINE

Compiler-Direktiven, Delphi 1055

\$DENYPACKAGEUNIT

Compiler-Direktiven, Delphi 1056

\$DENYPACKAGEUNIT '%s' kann nicht in ein Package übernommen werden (E2223) 1101

\$DESCRIPTION

Compiler-Direktiven, Delphi 1056

\$DESIGNONLY

Compiler-Direktiven, Delphi 1056

\$DESIGNONLY und **\$RUNONLY** sind nur in einer Package-Unit erlaubt (E2224) 1162

\$ELSE

Compiler-Direktiven, Delphi 1057

\$ELSEIF

Compiler-Direktiven, Delphi 1057

\$ENDIF

Compiler-Direktiven, Delphi 1057

\$EXTENDEDNTAX

Compiler-Direktiven, Delphi 1058

\$EXTENSION

Compiler-Direktiven, Delphi 1058

\$EXTERNASYM

Compiler-Direktiven, Delphi 1059

\$EXTERNASYM und **\$NODEFINE** sind für '%s' nicht erlaubt; nur globale Symbole (E2240) 1242

\$FINITEFLOAT

Compiler-Direktiven, Delphi 1059

\$HINTS

Compiler-Direktiven, Delphi 1060

\$HPPEMIT

Compiler-Direktiven, Delphi 1060

\$HPPEMIT '%s' wird ignoriert (W1034) 1157

\$IF

Compiler-Direktiven, Delphi 1061

\$IFDEF

Compiler-Direktiven, Delphi 1060

\$IFEND

Compiler-Direktiven, Delphi 1062

\$IFDEF

Compiler-Direktiven, Delphi 1062

\$IFOPT

Compiler-Direktiven, Delphi 1063

\$IMAGEBASE

Compiler-Direktiven, Delphi 1063

\$IMPLICITBUILD

Compiler-Direktiven, Delphi 1064

\$IMPORTEDDATA

Compiler-Direktiven, Delphi 1064

\$INCLUDE

Compiler-Direktiven, Delphi 1064

\$IOCHECKS

Compiler-Direktiven, Delphi 1065

\$LIB

Compiler-Direktiven, Delphi 1065

\$LINK

Compiler-Direktiven, Delphi 1066

\$LOCALSYMBOLS

Compiler-Direktiven, Delphi 1066

\$MESSAGE

Compiler-Direktiven, Delphi 1067

\$METHODINFO

Compiler-Direktiven, Delphi 1067	Compiler-Direktiven, Delphi 1074
\$METHODINFO-Direktive (Delphi) 1067	\$UNSAFECODE
\$MINENUMSIZE	Compiler-Direktiven, Delphi 1074
Compiler-Direktiven, Delphi 1068	\$VARSTRINGCHECKS
\$NODEFINE	Compiler-Direktiven, Delphi 1074
Compiler-Direktiven, Delphi 1070	\$WARN
\$NOINCLUDE	Compiler-Direktiven, Delphi 1075
Compiler-Direktiven, Delphi 1070	\$WARNINGS
\$ObjExportAll	Compiler-Direktiven, Delphi 1076
Compiler-Direktiven, Delphi 1058	\$WEAKPACKAGEUNIT
\$OPENSTRINGS	Compiler-Direktiven, Delphi 1076
Compiler-Direktiven, Delphi 1068	\$WEAKPACKAGEUNIT '%s' darf keinen Initialisierungs- oder Finalisierungscode enthalten (E2221) 1262
\$OPTIMIZATION	\$WEAKPACKAGEUNIT '%s' enthält globale Daten (E2203) 1262
Compiler-Direktiven, Delphi 1068	\$WEAKPACKAGEUNIT & \$DENYPACKAGEUNIT wurden beide angegeben (E2222) 1100
\$OVERFLOWCHECKS	\$WRITEABLECONST
Compiler-Direktiven, Delphi 1069	Compiler-Direktiven, Delphi 1077
\$RANGECHECKS	%
Compiler-Direktiven, Delphi 1070	%s '%s' doppelt mit identischen Parametern; Zugriff von C++ nicht möglich (W1029) 1132
\$REALCOMPATIBILITY	%
Compiler-Direktiven, Delphi 1071	'%s' Anweisung im OLE Automatisierungsbereich nicht erlaubt (E2182) 1162
\$REGION; \$ENDREGION	%
Compiler-Direktiven, Delphi 1071	%s erwartet, aber %s gefunden (E2029) 1140
\$RESOURCE	%
Compiler-Direktiven, Delphi 1072	'%s' ist kein Name einer Unit (E2242) 1212
\$RESOURCERESERVE	'%s' ist kein Typenbezeichner (E2005) 1247
Compiler-Direktiven, Delphi 1078	%
\$RUNONLY	%s kann nicht auf ein rechteckiges, dynamisches Array angewendet werden (E2432) 1181
Compiler-Direktiven, Delphi 1072	
\$SAFEDIVIDE	
Compiler-Direktiven, Delphi 1069	
\$SetPEFlags	
Compiler-Direktiven, Delphi 1078	
\$STACKFRAMES	
Compiler-Direktiven, Delphi 1077	
\$TYPEDADDRESS	
Compiler-Direktiven, Delphi 1073	
\$TYPEINFO	
Compiler-Direktiven, Delphi 1073	
\$UNDEF	

<KeyImpl> 472
<SingleLinkDef> 473

'%s' kann nicht mehrfach exportiert werden (E2287) 1193

'%s' muss eine Eigenschaft oder ein Feld der Klasse '%s' referenzieren (E2292) 1230

'%s' wurde vorher nicht als PROPERTY deklariert (E2174) 1181

%

%-Abschnitt ist nur in Klassentypen gültig (E2184) 1162

%s-Klausel erwartet, aber %s gefunden (E2128) 1110

.NET VCL-Komponenten 288

.NET-Assemblierungen 475

'; nicht erlaubt vor einem 'ELSE' (E2153) 1240

@

@-Operator

Operatoren 877

<

<AliasDef> 460

<AttributeDef> 461

<ClassDef> 462

<Classes> 463

<ConstantColumn> 464

<dateiname.PAS> oder <dateiname.DCU> kann im aktuellen Suchpfad nicht gefunden werden 364

<DiscriminatorColumn> 465

<DiscriminatorDef> 466

<DiscriminatorImpl> 467

<DiscriminatorValue> 468

<Globals> 469

<IDname> ist kein gültiger Bezeichner 356

<KeyColumn> 470

<KeyDef> 471

1

16-Bit Fixup in Objektdatei '%s' entdeckt (E2103) 1147

16-Bit-Segment in Objektdatei '%s' entdeckt (E2215) 1240

4

486/487-Instruktionen sind nicht aktiviert (E2117) 1088

A

A (Anker) HTML-Element 426

Ablaufsteuerung 1030

Abschnitt '%s' in Objektdatei '%s' kann nicht behandelt werden (E2216) 1252

Abschnitte (Delphi und C#) 1071

Abschnittssteuerelemente

anzeigen 213

ABSTRACT und FINAL dürfen nicht zusammen verwendet werden (E2371) 1085

ABSTRACT und SEALED dürfen nicht zusammen verwendet werden (E2383) 1086

Abstrakte Methoden müssen virtuell oder dynamisch sein (E2167) 1086

Active Form-Experte 370

Active-Server-Objekt-Experte 371

Adresse eingeben 677

Adresse von lokalem Symbol %s kann nicht aufgenommen werden (E2278) 1102

Aktionen in einer Aktivität gruppieren 164

Aktionslisten-Editor 613

Aktionsmanager-Editor 611

Aktive Sprache festlegen 484

Aktive Sprache für ein Projekt festlegen 90

Aktivierungsleiste 1305

Aktivität für einen Zustand erzeugen 148

Aktivitätsdiagramm

Beispiel 1386

Aktivitätsparameter

hinzufügen 162

Aktualisierung durchführen	724	Array-Parameter
Alle Anleitungen	1	Parameter 938
Allgemeines zu Together	1380	Arrays 902
An '%s' zugewiesener Wert wird niemals benutzt (H2077)	1258	Array-Typ erforderlich (E2016) 1193
Anderes Symbol	604	Artefakt
		verteilen 174
Ä		Arten von Klassendiagrammen 1295
Änderung	657	ASP .NET 495
Änderungen in der Pattern-Registrierung speichern	244	ASP.NET 725
Änderungsbalken		ASP-Deployment-Manager 479
Quelltext-Editor	1373	Assembler-Ausdrücke
		Integrierter Assembler 1040
A		Assembler-Ausdrücke (nur Win32) 1040
Andocken		Assembler-Prozeduren und -Funktionen
Tools	66	Integrierter Assembler 1048
Anfangswert		Assembler-Prozeduren und -Funktionen (nur Win32) 1048
definieren	216	Assembler-Syntax
Anleitungen	1389	Integrierter Assembler 1035
Ansichtsfilter verwenden	194	Assembler-Syntax (nur Win32) 1035
Anweisung erforderlich, aber Ausdruck vom Typ '%s' gefunden (E2014)	1205	Assemblierung '%s' konnte nicht importiert werden, weil sie den Namespace '%s' enthält (E2415) 1182
Anweisungen sind im Interface-Teil nicht erlaubt (E2050)	1099	Assemblierungs-Metadaten-Explorer 68, 69, 833
Anweisungstext-Editor	299, 300	Assemblierungsübergreifende protected-Referenz auf [%s]%.%s in %.%s (E2364) 1184
Anwenden der aktiven Build-Konfiguration auf ein Projekt	2	Assert-Direktiven (Delphi) 1053
Anwendung	489, 490	assignment statements
Anwendung (Visual Basic)	491	statements 862
Anwendungen debuggen	1439	Assoziationsklasse
Anwendungen lokalisieren	1437	löschen 208
Anwendungen mit Together modellieren	1446	Assoziationsklasse erstellen 208
Anwendungen testen	1425	Assoziationsklassen und n-fache Assoziationen 1298
Anwendungstyp (Delphi)	1053	ATL 525
Anzahl der Elemente (%d) ist von der Deklaration (%d) unterschiedlich (E2072)	1170	Attribut - Bekanntes Attribut darf keine Eigenschaften festlegen (E2313) 1105
Anzeigeoptionen	738	Attribut - Bekanntes benutzerdefiniertes Attribut für ungültiges Ziel (E2315) 1105
Arbeiten mit benannten Optionsgruppen	15	Attribut - Bekanntes benutzerdefiniertes Attribut hatte ungültigen Wert (E2317) 1106
Arbeitsblatt-Editor	637	Attribut - Bekanntes benutzerdefiniertes Attribut wiederholt das benannte Argument (E2321) 1106
Arithmetische Operatoren		Attribut - Benanntes Argument des bekannten Attributs unterstützt keine Varianten (E2324) 1106
Operatoren	877	
Array-Eigenschaften		
Eigenschaften	965	

Attribut - Benanntes Argument eines bekannten Attributs darf kein Array sein (E2309) 1105	Auf Variable '%s' kann wegen Optimierung nicht zugegriffen werden (E2171) 1260
Attribut - Das Attribut MarshalAs hat Felder, die für den angegebenen unverwalteten Typ nicht zulässig sind (E2314) 1105	Auf Ziel kann nicht zugegriffen werden (E2144) 1126
Attribut - Das Format der GUID war ungültig (E2316) 1105	Aufgrund des Hard-Mode konnte nicht gestoppt werden 343
Attribut - Der angegebene unverwaltete Typ ist nur für Felder gültig (E2320) 1106	Auflistungs-Editor 456, 616
Attribut - Die Konstantengröße von MarshalAs darf nicht negativ sein (E2318) 1106	Aufruf der abstrakten Methode %s.%s (E2373) 1085
Attribut - Die Parametergröße von MarshalAs darf nicht negativ sein (E2319) 1106	Aufrufkonventionen
Attribut - Für den festen MarshalAs-String ist eine Größenangabe erforderlich (E2311) 1105	Prozeduren und Funktionen 931
Attribut - Für einen benutzerdefinierten Marshaler ist ein benutzerdefinierter Marshaler-Typ erforderlich (E2310) 1105	Aufruf-Stack 785
Attribut - Unerwarteter Typ in bekanntem Attribut (E2322) 1106	Aufzählungstypen
Attribut - Unerwartetes Argument für ein bekanntes Attribut (E2323) 1106	Datentypen 889
Attribut - Ungültiges Argument für ein bekanntes Attribut (E2312) 1105	Aus %s kann kein eindeutiger Typ erstellt werden (E2374) 1101
Attribut '%s' ist für dieses Ziel nicht gültig (E2325) 1104	Aus dem Projekt entfernen 597
Attribut '%s' kann nur einmal pro Ziel verwendet werden (E2326) 1104	Ausdruck hat keinen Wert (E2143) 1141
Audit 1456	Ausdruck hinzufügen 17
Audit drucken (Dialogfeld) 698	Ausdruck zu komplex (E2156) 1244
Audit-Ergebnis (Fenster) 1273	Ausdrücke 877
Audit-Ergebnis speichern (Dialogfeld) 706	Ausdruckstyp muss BOOLEAN sein (E2012) 1194
Audit-Ergebnisse anzeigen 270	Ausdruckstyp muss BOOLEAN sein (E2012) (E2013) 1197
Audit-Ergebnisse drucken 268	Ausführungs- und Aufrufspezifikationen 1320
Audit-Ergebnisse exportieren 267	Ausführungs- und Aufrufspezifikationen kopieren und einfügen 176
Audits	Ausgabedatei '%s' kann nicht erstellt werden (F2039) 1144
gruppieren 270	Auslösen und Behandeln von
im Quelltext navigieren 270	Exceptions 977
sortieren 270	Ausrichtung 336
Audits ausführen 269	Auswahlmanager 709
Audits für "Cash Sales" ausführen 1383	Auswerten
Auf Eigenschaft '%s' kann hier nicht zugegriffen werden (E2233) 1229	Ändern/debuggen 1439
Auf Element 0 kann nicht zugegriffen werden - 'Length' oder 'SetLength' verwenden (E2157) 1257	Auswerten/Ändern 661
Auf private-Symbol %s.%s kann nicht zugegriffen werden (E2361) 1236	AUTOBOX-Direktive (Delphi für .NET) 1053
Auf protected-Symbol %s.%s kann nicht zugegriffen werden (E2362) 1234	AutoFormat 455
	Automatisches Layout für Diagramme 127
	Automatisierungsobjekte
	Schnittstellen 1015
	Automatisierungsobjekte (nur Win32) 1015

B

Balkendiagramm

Balkendiagramm 1456

Bearbeiten der Hyperlinks für Diagramm (Dialogfeld) 686	Bibliotheken und Packages 997
Bearbeitungs-Tool 717	Bibliotheksname zu lang: %s (E2286) 1121
Bedingungsausdrücke 1307	Bibliotheksverwaltung 576
Bedingungssammlungs-Editor 296	Bild einfügen 445
Befehlszeilenoptionen 1333	Bild laden (Dialogfeld) 634
Bei der Compilierung zu Byte-Code können absolute Variablen nicht benutzt werden (E2249) 1084	Bild speichern unter 641
Bei der Compilierung zu Byte-Code können Objekttypen alten Stils nicht benutzt werden (E2248) 1219	Bild-Editor 640
Bei der vorherigen Deklaration von '%s' wurde die Direktive 'overload' nicht angegeben (E2267) 1201	Bilderlisten-Editor 621
Bei Umwandlung der angegebenen WideString-Konstante gehen Informationen verloren (H2455) 1121	blocks 862
Beispiel für das Suchen von Referenzen 1421	blocks and scope 862
Beispielprojekt "Cash Sales" 1381	Blockvervollständigung 1373
Benachrichtigung über Debugger-Exception 658	Boolesche Kurzauswertung (Delphi-Compiler-Direktive) 1054
Benutzerdefinierte Attribute 1049	Boolesche Operatoren
Benutzerdefinierte Attribute in .NET 1049	Operatoren 877
Benutzerdefinierte Komponenten installieren 70	Boolesche Typen
Benutzereigenschaften hinzufügen/entfernen (Dialogfeld) 683	Datentypen 889
Benutzereigenschaften verwenden 135	Borland.Delphi.Compiler.ResCvt.dll konnte nicht gefunden werden (E2377) 1239
Benutzeroberflächen entwerfen 1410	Botschaftsmethoden
Bereich für Exception hinzufügen 722	Methoden 956
Bereichsüberprüfung 1070	BREAK oder CONTINUE außerhalb der Schleife (E2097) 1100
Beschreibung 511	BREAK, CONTINUE oder EXIT bei FINALLY-Klausel nicht möglich (E2126) 1103
Beschreibung (Delphi) 1056	Build-Ausgabe 1461
Betriebssystemfehler 1083	Build-Ereignisse 497
Bezeichner '%s' kann nicht exportiert werden (E2276) 1100	Build-Konfigurationen 1351, 1461, 1463, 1465
Bezeichner (Delphi)	MSBuild 2
Zeichensätze (Delphi) 857	Build-Konfigurationen, aktive 1463, 1465
Bezeichner erwartet (E2372) 1158	
Bezeichner redeklariert: '%s' (E2004) 1158	
Beziehung 321	C
Beziehungen in Klassendiagrammen 1296	C++ Compiler Ausgabe 545
Beziehungen nach Pattern erstellen 226	C++ Compiler Debugging 534
Beziehungen umlenken 130	C++-Compiler 543
Beziehungskollektions-Editor 297	C++-Compiler Allgemeine Compilierung 538
Beziehungslinien mit Umlenkpunkten erstellen 137	C++Compiler C++-Compilierung 531
Bibliothek 737	C++-Compiler C++-Kompatibilität 529, 536
Bibliotheken	C++-Compiler Erweiterte Compilierung 526
Initialisierungscode 998	C++-Compiler Optimierungen 544
	C++-Compiler Pfade und Definitionen 546
	C++-Compiler Vorcompilierte Header 547

C++-Compiler Warnungen 549

Caliber

Anforderungen 1406

case statements 862

Case-Label außerhalb des Bereichs des Case-Ausdrucks (W1018) 1103

Cassini konfigurieren 493

Chmod-Fehler für '%s' (x2044) 1144

Close (Funktion)

Gerätetreiber 989

CLS: Überschreiben der virtuellen Methode '%s.%s'. Sichtbarkeit ('%s') muss mit der Basisklasse '%s' ('%s') übereinstimmen (H2384) 1111

Code Insight 48, 728

Quelltext-Editor 1373

Code Insight verwenden 48

Code-Explorer 776

Code-Folding 42

Code-Folding verwenden 42

CodeGear-Debugger 726

CodeGuard 498

CodeGuard-Konfiguration 712

Codepage '%s' ist auf diesem Rechner nicht installiert (E2424) 1178

Code-Visualisierungsdiagramm 278

COM+-Subskriptionsobjekt-Experte 379

COM-Importe 474

COM-Objekt-Experte 376

Compiler 499, 501

Compiler (Visual Basic) 508

Compiler-Meldungen 507

Compilieren und Builds erstellen – Anleitungen 1391

Compilieren von Entwurfszeit-Packages, die Delphi-Quelltextdateien enthalten 4

Compilieren, Build erstellen und Anwendungen ausführen 1351

Compilierung durch Anwender abgebrochen (E2165) 1255

compound statements 862

CPU-Fenster 786

CREATE erwartet (E2412) 1120

D

Das Eigenschaftsattribut 'label' darf nicht leer bleiben (E2275) 1230

Das Eigenschaftsattribut 'label' kann in dispinterface nicht verwendet werden (E2274) 1128

Das Erscheinungsbild von Abschnitten ändern 213

Das Erscheinungsbild von Interfaces ändern 214

Das Feld <feld> hat keine korrespondierende Komponente.Deklaration entfernen? 349

Das Feld <feld> sollte vom Typ <typ1> sein, ist jedoch als <typ2> deklariert. Deklaration korrigieren? 350

Das Interface '%s' wird in der Liste der Interfaces nicht erwähnt (E2265) 1168

Das private-Symbol '%s' wurde deklariert, aber nie verwendet (H2219) 1226

Das Profil "UML in Farbe" verwenden 119

Das Programm oder die Unit %s ruft sich selbst wieder auf (F2092) 1237

Das Projekt enthält bereits ein Formular bzw. ein Modul mit dem Namen <name> 347

Das Projekt zum Debuggen vorbereiten 27

Das Published-Feld '%s' ist weder vom Typ class noch interface (E2217) 1101

Das Überschreiben von Eigenschaften ist im Typ interface nicht erlaubt (E2206) 1234

Das UML 2.0-Beispielprojekt öffnen 1384

Datei einbinden (Delphi) 1064

Datei nicht gefunden 800

Datei nicht gefunden: '%s' (x1026) 1144

Datei-Browser 806

Dateiein- und -ausgabe

Standardroutinen 989

Dateien für das externe Debuggen vorbereiten 35

Dateien hinzufügen und entfernen 57

Dateien in der Projektverwaltung umbenennen 71

Dateiname zu lang (übersteigt %d Zeichen) (E2288) 1144

Dateityp ist hier nicht zulässig (E2002) 1145

Dateitypen 902

Datenadapter konfigurieren 301

Datenadapter-Konfiguration DataSet 305

Datenadapter-Konfiguration Daten in der Vorabansicht 306

Datenbank	Definition eines UML 1.5-Kollaborationsdiagramms 1303
neue Verbindung hinzufügen 80	Definition eines UML 1.5-Komponentendiagramms 1310
Datenbank-Editor 307	Definition eines UML 1.5-Sequenzdiagramms 1302
Datenbankevolution 328	Definition eines UML 1.5-Verteilungsdiagramms 1311
Datenbankformular-Experte 308	Definition eines UML 1.5-Zustandsdiagramms 1305
Datenbankkomponenten-Editor (Dialogfeld) 315	Definition eines UML 2.0-Aktivitätsdiagramms 1324
Datenbindungen 457	Definition eines UML 2.0-Klassendiagramms 1313
Daten-Explorer 778	Definition eines UML 2.0-Kommunikationsdiagramms 1316
Datenmenge erzeugen 314	Definition eines UML 2.0-Komponentendiagramms 1326
Datentyp zu groß: 2 GB überschritten (E2100) 1248	Definition eines UML 2.0-Kompositionssstrukturdiagramms 1329
Datentypen, Variablen und Konstanten 887	Definition eines UML 2.0-Sequenzdiagramms 1316
DDE-Info (Dialogfeld) 607	Definition eines UML 2.0-Verteilungsdiagramms 1328
Deaktivieren von Themes in der IDE und in Ihrer Anwendung 65	Definition eines UML 2.0-Zustandsmaschinendiagramms 1321
Debug-Desktop einstellen 815	Definition für abstrakte Methode '%s' nicht erlaubt (E2136) 1085
Debuggen	Deklaration der Klasse <klassenname> fehlt oder ist falsch 351
Ausdruck hinzufügen 17	Deklaration von '%s' unterscheidet sich von der Deklaration in interface '%s' (E2211) 1175
Ausdrücke bearbeiten 26	Deklaration von '%s' unterscheidet sich von vorheriger Deklaration (E2037) 1148
Datenelemente untersuchen 24	Deklarierte Konstanten 926
externe Anwendungen 28, 32	Dekomposition
Haltepunkte 19	definieren 179
Projekt vorbereiten 22, 27	Delegation, Schnittstellen implementieren durch Schnittstellen 1009
Prozess verbinden 18	Delegationskonnektor 1331
Übersicht 1439	Delegationskonnektoren erstellen 168
Debugger 510	Delphi Compiler-Fehler 1079
Debugger auf einem externen Computer installieren 30	Delphi-Compiler 559
Debugger-Optionen 735	Delphi-Compiler Compilierung 553
Debugger-Symboltabellen 522	Delphi-Compiler Pfade und Definitionen 562
Debugger-Umgebung 513	Delphi-Compiler Warnungen 563
Debugging – Anleitungen 1392	Delphi-Compiler Weitere Optionen 560
Debug-Informationen (Delphi) 1055	Delphi-Referenz 842
Debug-Inspektor 659	Delphi-Sprache 842
declarations 862	Delphi-Sprachreferenz 842
declarations and statements 862	Delphi-Units in eine Anwendung linken 10
Declarations and Statements 862	Dem Wörterbuch hinzufügen 807
DEFINE-Direktive (Delphi) 1055	Den Pattern-Organizer verwenden 246
Definition des UML 1.5-Anwendungsfalldiagramms 1300	Den Typ einer Beziehung ändern 121
Definition des UML 2.0-Anwendungsfalldiagramms 1314	DENYPACKAGEUNIT-Direktive (Delphi) 1056
Definition eines UML 1.5-Aktivitätsdiagramms 1309	
Definition eines UML 1.5-Klassendiagramms 1294	

Deployment	in Projekt verschieben 272
Übersicht 1443	umbenennen 129
Deployment von Anwendungen 41, 1443	Diagramm als Bild exportieren 279
Deployment von ASP.NET-Anwendungen 40	Diagramm als Bild exportieren (Dialogfeld) 687
Deploymentmanager 383	Diagramm drucken (Dialogfeld) 699
Der Ausdruck benötigt kein Initialize/Finalize (x2243) 1247	Diagrammansicht 1269
Der Debugger läuft gerade. Beenden? 367, 670	öffnen 262
Der Implements-Getter darf keine dynamische oder message-Methode sein (E2263) 1165	Diagrammdateien der Versionskontrolle unterstellen 144
Der Implements-Getter muss die Aufrufkonvention %s haben (E2262) 1167	Diagramme
Der linken Seite kann nichts zugewiesen werden (E2064) 1212	speichern 272
Der Name des Schlüsselcontainers '%s' ist nicht vorhanden (E2387) 1181	Diagramme als Grafik exportieren 136
Der Objektablage hinzufügen 477	Diagramme drucken 143
Der Objektablage hinzufügen 775	Diagramme erstellen 116
Der reservierte Unit-Name '%s' kann nicht verwendet werden (E2272) 1244	Diagramme löschen 124
Der Vergleich ergibt immer False (W1021) 1112	Diagramme mit Hinweisen versehen 115
Der Vergleich ergibt immer True (W1022) 1113	Diagramme schließen 122
Der voll qualifizierte, verschachtelte Typname %s übersteigt die 1024 Byte-Grenze (E2409) 1206	Diagramme umbenennen 129
Der Vorfahrtyp '%s' besitzt keinen zugreifbaren Standardkonstruktor (E2253) 1191	Diagramme zoomen 195
Der Vorgabeparameter '%s' muss als Wert oder Konstante übergeben werden (E2239) 1123	Diagrammformat – Überblick 1449
Der Zeigerausdruck benötigt kein Initialize/Finalize, aber möglicherweise einen ^-Operator (H2244) 1235	Dialogfelder zur Elementauswahl 708
Designer-Richtlinien für VCL-Komponenten verwenden 80	Dialogfenster für COM+ Ereignisschnittstelle-Auswahl 377
DESIGNONLY-Direktive (Delphi) 1056	Die <methodenname>, aufgerufen von <formularname>. <ereignisname> hat eine inkompatible Parameterliste. Soll der Aufruf entfernt werden? 360
Designprojekte in Quelltext umwandeln 264	Die Bibliothek <bibliothek> ist schon geladen, wahrscheinlich aufgrund eines ungültigen Programmabbruchs. Ihr System könnte instabil sein, deshalb sollten Sie Windows jetzt beenden und neu starten. 357
Desktop speichern 814	Die Compilierung wurde wegen zu vieler Fehler abgebrochen (E2226) 1245
Desktop-Layouts speichern 72	Die Direktive '%s' ist im Typ dispinterface nicht erlaubt (E2231) 1163
Destruktoren	Die Direktive '%s' ist im Typ interface nicht erlaubt (E2210) 1126
Methoden 956	Die Erzeugung der für die Eigenschaft %s.%s erforderliche(n) Zugriffsmethode(n) ist wegen eines Namenskonflikts mit einem vorhandenen Symbol in demselben Gültigkeitsbereich nicht möglich (E2392) 1102
Destruktoren können nicht mit IDisposable gemischt werden (E2290) 1126	Die exportierte Package-Thread-Variable '%s.%s' darf nicht außerhalb dieses Package verwendet werden (W1032) 1224
Diagramm	Die IDE für das Simulieren von Delphi 7 einrichten 78
aus Versionskontrolle ausschließen 144	Die Klausel '%s' ist im Typ interface nicht erlaubt (E2207) 1164
ein/auschecken 144	Die Methode <methodenname>, aufgerufen von <formularname>, existiert nicht. Soll der Aufruf entfernt werden? 361
erstellen 272	
in Bild exportieren 272	

Die Online-Hilfe verwenden 114	Doppelte Implementierung für 'set of %s' in diesem Gültigkeitsbereich (E2429) 1138
Die Packages '%s' und '%s' enthalten beide Unit '%s' (E2199) 1224	Doppelte implements-Klausel für Interface '%s' (E2257) 1134
Die Pattern-Registrierung verwenden 247	Doppelte Ressourcen-ID: Typ %d ID %d (E2285) 1138
Die Published-Real-Eigenschaft '%s' muss vom Typ Single, Real, Double oder Extended sein (E2186) 1235	Doppelte Symbolnamen im Namespace. '%s.%s' in %s gefunden. Duplikat in %s wird ignoriert (W1051) 1131
Die Schreibweise der überschriebenen Methode %s.%s sollte mit der des Vorfahren %s.%s übereinstimmen (H2365) 1222	Doppelten Ressourcenbezeichner %s in Unit %s(%s) und %s(%s) gefunden (E2407) 1138
Die Suchreihenfolge für Debug-Symboltabellen festlegen 36	Doppelter Botschaftsmethoden-Index (E2140) 1135
Die Typen der tatsächlichen und formalen Var-Parameter müssen übereinstimmen (E2033) 1260	Doppelter Ressourcename: Typ %d '%s' (E2284) 1138
Die überladene Prozedur '%s' muss mit der Direktive 'overload' gekennzeichnet sein (E2254) 1200	Doppelter Tag-Wert (E2027) 1138
Die Unit '%s' wurde implizit in Package '%s' importiert (x1033) 1169	Doppeltes Case-Label (E2030) 1131
Die Uses-Klausel fehlt oder ist falsch. 368	Doppeltes Symbol '%s' im Namespace '%s' von '%s' und '%s' definiert (E2447) 1135
Diese Form des Methodenaufrufs ist nur für Klassenmethoden erlaubt (E2076) 1195	Drag&Drop 134
Diese Form des Methodenaufrufs ist nur in Methoden von abgeleiteten Typen erlaubt (E2075) 1198	Druckauswahl 397
Dieser Typ kann nicht initialisiert werden (E2071) 1101	Durch das Überschreiben erhält die virtuelle Methode '%s.%s' eine geringere Sichtbarkeit (%s) als die Basisklasse '%s' (%s) (x2269) 1207
Direktiven (Delphi) 857	Durch Umwandlung der angegebenen WideChar-Konstante (#\$%04X) auf AnsiChar gehen Informationen verloren (H2451) 1121
Direktiven für Bibliotheken und gemeinsame Objekte (Delphi) 1065	Durchsuchen (Dialogfeld) 603
dispid '%d' wird bereits von '%s' verwendet (E2180) 1133	Dynamisch ladbare Bibliotheken
dispid-Klausel nur im OLE-Automatisierungsbereich erlaubt (E2183) 1127	Bibliotheken 998
Dispose	Dynamisch ladbare Bibliotheken schreiben 998
Delphi für .NET 1027	Dynamisch zugewiesene, multidimensionale Arrays
Dispose wird für dynamische Arrays nicht unterstützt (da nicht erforderlich) (E2256) 1213	Arrays 902
Disposed_ darf nicht in Klassen mit Destruktoren deklariert werden 1265	Dynamische Arrays
Disposed_ darf nicht in Klassen mit Destruktoren deklariert werden (E2414) 1128	Arrays 902
DIV HTML-Element 429	Dynamische Eigenschaften 74, 458
Division durch Null (E2098) 1131	Dynamische Eigenschaften festlegen 74
Division durch Null (E2118) 1089	Dynamische oder Botschaftsmethoden sind hier nicht erlaubt (E2148) 1228
DLLs, aufrufen 997	Dynamische und Botschaftsmethoden sind im OLE-Automatisierungsbereich nicht erlaubt (E2178) 1139
Dok-Kommentare 1451	Dynamischer Array-Typ erforderlich (E2413) 1139
Dokumentation erzeugen (Dialogfeld) 690	
Dokumentation für "Cash Sales" erzeugen 1383	
Dokumentationserzeugung 1458	
Doppeldeutiger überladener Aufruf von '%s' (E2251) 1086	

E

E

 A-Fehler 1083

 E/A-Fehler 1083

 E/A-Prüfung (Delphi) 1065

ECO - Allgemeine Optionen 739	Ein Typ dispinterface kann keinen Vorfahr interface haben (E2228) 1129
EcoSpace-Designer 329	Eindeutige Einschränkung 327
Editor für Verbindungs-Strings (ADO) 304	Eine Adresse kann bei der Compilierung zu Byte-Code nicht akzeptiert werden (E2247) 1086
Editor-Optionen 741	Eine andere Datei mit dem Namen <dateiname> befindet sich bereits im Suchpfad 342
Eigenschaft '%s' existiert nicht in Basisklasse (E2147) 1248	Eine Anwendungsfallhierarchie erstellen 222
Eigenschaft erforderlich (E2299) 1202	Eine benötigte Bibliothekshilfsfunktion wurde vom Linker eliminiert (%s) (E2172) 1153
Eigenschaft und Methode <methodename> sind nicht kompatibel 363	Eine Diagrammabfolge erstellen 220
Eigenschaften 459, 965	Eine einfache Beziehung erstellen 141
Klassen und Objekte 965	Eine final-Methode kann nicht überschrieben werden (E2352) 1223
Eigenschaften bearbeiten 242	Eine Implementation der Methode <methodename> kann nicht gefunden werden 359
Eigenschaften Darstellung überwachter Ausdrücke 672	Eine interne Struktur für einen Knoten erzeugen 169
Eigenschaften der Datenmenge 309	Eine Komponentenklasse mit dem Namen <name> existiert bereits 345
Eigenschaften überschreiben	Eine nicht-virtuelle Methode kann nicht überschrieben werden (E2170) 1223
Eigenschaften 965	Eine unterstützende Klasse darf keinen Destruktor einführen (E2295) 1153
Eigenschaften und Ereignisse festlegen 77	Eine von '%s' abgeleitete, unterstützende Klasse kann nur Klassen unterstützen, die von '%s' abstammen. (E2294) 1181
Eigenschaftsdeklaration verweist auf private-Vorfahr '%s.%s' (W1045) 1226	Einem Container ein Member hinzufügen 212
Eigenschaftsdeklarationen sind im anonymen Record- oder lokalem Record-Typ nicht zulässig (E2434) 1228	Einer Nur-Lesen Eigenschaft kann kein Wert zugewiesen werden (E2129) 1230
Eigenschaftszugriff 965	Einfache Typen 889
Eigenschaftszugriffsmethode '%s' kann nicht erzeugt werden, weil '%s' bereits vorhanden ist (E2300) 1232	Datentypen 889
Eigenschaftszugriffsmethode '%s' sollte %s sein (H2369) 1232	Einführende Anleitungen 1390
Ein Attribut-Argument muss ein konstanter Ausdruck, ein typeof-Ausdruck oder ein Array-Konstruktor sein (E2448) 1094	Einführung 1353
Ein DLLImport-Attribut und eine externe oder Aufrufkonventionsdirektive dürfen nicht zusammen verwendet werden (E2293) 1131	Benutzerdefinierte Komponenten installieren 70
Ein erster Blick auf die IDE 1363	Datei-Browser 81
Ein Feld oder eine Methode mit dem Namen <name> existiert bereits 346	Dateien umbenennen 71
Ein in einer unterstützenden Klasse eingeführter Konstruktor muss den parameterlosen Konstruktor der unterstützten Klasse als erste Anweisung aufrufen. (E2296) 1180	Dateien zu einem Projekt hinzufügen 57
Ein NeverBuild-Package '%s' benötigt ein AlwaysBuild-Package '%s' (E2220) 1207	Desktop-Layouts speichern 72
Ein Pattern erstellen 231	Eigenschaften und Ereignisse festlegen 77
Ein Projekt erstellen 61, 249	Ereignisbehandlungs Routinen schreiben 85
Ein Projekt starten 1369	Formulare anpassen 62
Ein referenziertes Part erstellen 171	Komponenten suchen 67
Ein Symbol umbenennen 11	Komponenten zu Formularen hinzufügen 55
Ein Typ dispinterface benötigt eine interface-Identifikation (E2229) 1131	Komponenteneigenschaften festlegen 73

Projekte erstellen 61	Entwickeln von Tests 1427
Projektoptionen einstellen 75, 76	Entwicklungszyklus
Referenzen kopieren 59	Tool-Überblick 1406
Symbolleisten anordnen 63	Entwicklungszyklus steuern 1405
Templates zur Objektablage hinzufügen 58	Entwurfs-Packages hinzufügen 486
Tool-Fenster andocken 66	Entwurfspattern 1454
Tool-Palette anpassen 64	Ereignisbehandlungs routinen schreiben 85
Vista-Themes 65	Ereigniseigenschaften und Ereignisbehandlungs routinen
Voreinstellungen für Tools festlegen 79	Ereignisse 971
Eingabe einfügen 446	Ereignisprotokoll-Optionen 745
Eingabemasken-Editor 628	Ereignisse 971
Eintritts- und Austrittsaktion erstellen 160	Klassen und Objekte 971
Eintritts- und Austrittsaktionen festlegen 204	Erforderliches Interface
Einzelnes Modellelement erstellen 142	erstellen 207
EJBs aus Liste auswählen 449	Ergebnisse löschen
Elemente der Tool-Palette suchen 67	Refactoring löschen 1423
Elemente in einem UML 1.5-Aktivitätsdiagramm 1308	Erste Schritte mit Together 1380
Elemente in einem UML 1.5-Anwendungsfalldiagramm 1301	Erstellen von benannten Build-Konfigurationen für C++ 6
Elemente in einem UML 1.5-Interaktionsdiagramm 1304	Erstellen von benannten Build-Konfigurationen für Delphi 7
Elemente in einem UML 1.5-Klassendiagramm 1296	Erstellen von Build-Ereignissen 5
Elemente in einem UML 1.5-Komponentendiagramm 1311	Erstellungsfolge 337
Elemente in einem UML 1.5-Verteilungsdiagramm 1312	Erweiterbare Kurzhinweise 23
Elemente in einem UML 1.5-Zustandsdiagramm 1306	Erweiterte Datenbindung 454
Elemente in einem UML 2.0-Aktivitätsdiagramm 1325	Erweiterte Informationen zu überwachten Ausdrücken anzeigen 39
Elemente in einem UML 2.0-Anwendungsfalldiagramm 1315	Erweiterte Syntax (Delphi) 1058
Elemente in einem UML 2.0-Klassendiagramm 1313	Erweiterung für ausführbare Dateien (Delphi) 1058
Elemente in einem UML 2.0-Kommunikationsdiagramm 1317	Erweiterungspunkt 1302
Elemente in einem UML 2.0-Komponentendiagramm 1327	Erweiterungspunkte erstellen 221
Elemente in einem UML 2.0-Kompositionssstrukturdiagramm 1330	Erzeugen eines Projekts mit einem MSBuild-Befehl 12
Elemente in einem UML 2.0-Sequenzdiagramm 1316	Es existiert keine überladene Version von '%s' mit dieser Parameterliste (E2273) 1214
Elemente in einem UML 2.0-Verteilungsdiagramm 1328	Es gibt bereits eine Methode '%s' mit identischen Parametern (E2252) 1136
Elemente in einem UML 2.0-Zustandsmaschinendiagramm 1322	Es gibt keine überladene Version von '%s', die man mit diesen Argumenten aufrufen kann (E2250) 1213
Element-Stereotyp zuweisen 133	Es kann nur entweder IID oder GuidAttribute angegeben werden (E2427) 1159
ELSE (Delphi) 1057	Es muss mindestens 1 Dimension für NEW des dyn. Array angegeben werden (E2308) 1087
ELSEIF (Delphi) 1057	Es muss mindestens 1 Dimension für SetLength des dyn. Array angegeben werden (E2246) 1087
ENDIF fehlt (E2095) 1190	
ENDIF-Direktive 1057	
Entfernen von FCC-Pattern aus dem Modell 232	

Es müssen C++-Obj.-Dateien generiert werden (-jp) (E2241)
1120

Es wurde keine Anweisung für die aktuelle Zeile generiert 362

Es wurden keine Konfigurationsdateien gefunden (W1039) 1213
EXCEPT oder FINALLY erwartet (E2125) 1140

Exceptions 977

 Klassen und Objekte 977

Exceptions erneut auslösen

 Exceptions 977

Exception-Standardklassen und -Standardroutinen

 Exceptions 977

Exception-Typen deklarieren

 Exceptions 977

Experte "VCL-Komponente importieren" 281

Experte für den WSDL-Import 420

Experte für Automatisierungsobjekte 374

Experte für COM+ Ereignisobjekte 378

Experte für externes Datenmodul 400

Experte für Interface-Auswahl 386

Experte für neue VCL-Komponenten 291

Experte für neues DBWeb Control 389

Experte für XML-Datenbindung, Seite 1 423

Experte für XML-Datenbindung, Seite 2 424

Experte für XML-Datenbindung, Seite 3 425

Experte zum Erstellen von Pattern 1278

Experte zum Kapseln von Datenbanken 334

Experte zum Konvertieren aus MDL 1276

Explorer 747

export-Klauseln

 Bibliotheken 998

EXPORTS nur im globalen Bereich erlaubt (E2191) 1141

EXPORTS-Abschnitte sind nur bei der Compilierung mit {\$UNSAFE CODE ON} zulässig (E2406) 1253

Externe Deklaration

 Prozeduren und Funktionen 931

Externe Symbole (Delphi) 1059

Externen Translation-Manager einrichten 91

Externen Translation-Manager verwenden 94

F

Falsche Adressverschiebung in Objektdatei '%s' entdeckt (E2104) 1097

Falsche Feld-Deklaration in der Klasse <klassenname> 348

Falsche globale Symboldefinition: '%s' in Objektdatei '%s' (x1028) 1094

Falsche GUID-Syntax (E2204) 1188

Falsche Methoden-Deklaration in der Klasse <klassenname> 358

Falsche oder beschädigte RLINK32.DLL (E2152) 1263

Falscher Argumententyp im Konstruktor des VariablenTyp array (E2150) 1097

Falsches Dateiformat: '%s' (x2141) 1144

Falsches Objektdateiformat: '%s' (E2045) 1094

Falsches Package-Unit-Format: %s.%s (E2213) 1224

Falsches Unit-Format: '%s' (F2048) 1250

Farbe 732, 761

Farbe auswählen 448

Farbeditor 605

Farben 730, 759

Fehlender Parametertyp (E2067) 1192

Fehlendes oder ungültiges bedingtes Symbol in der Anweisung '\$%s' (E2173) 1178

Fehler beim Erstellen des Berichts: <prozess> (<fehlercode>) 344

Fehler beim Konvertieren der Ressource %s (E2378) 1140

Fehler beim Konvertieren der Unicode-Zeichen in den länderspezifischen Zeichensatz. String wurde abgeschnitten. Ist die Umgebungsvariable LANG korrekt gesetzt? (W1041) 1249

Fehler beim Konvertieren des Strings '%s' in Unicode. String wurde abgeschnitten. Ist die Umgebungsvariable LANG korrekt gesetzt? (W1042) 1185

Fehler beim Laden von .NET Framework %s: %08X (E2401) 1110

Fehler beim Lesen von '%s' (x2041) 1145

Fehler beim Schließen von '%s' (x2043) 1144

Fehler beim Schreiben von '%s' (x2042) 1146

Fehler beim Signieren der Assemblierung (E2385) 1140

Fehler beim Suchen von '%s' (F2040) 1145

Fehler in numerischer Konstante (E2115) 1090

Fehleraddresse nicht gefunden 341

Fehlerbehebung an Modellen 265

Fehlermeldungen	1084	for-in-Anweisung arbeitet nicht mit Kollektionstyp '%s' (E2430)
Feld '%s' muss initialisiert werden - in CLS-konformen Wertetypen nicht zulässig (E2428)	1171	1212
Feld deklarieren		for-in-Anweisung arbeitet nicht mit Kollektionstyp '%s', weil '%s' kein Element für '%s' enthält oder darauf nicht zugegriffen werden kann (E2431)
Refactoring	1415	1111
Feld einführen (Dialogfeld)	692	Formular anpassen
Feld- oder Methodenbezeichner erwartet (E2168)	1196	62
Felddefinition nicht erlaubt in OLE-Automatisierungsbereich (E2175)	1142	Formular anzeigen
Felddefinition nicht erlaubt nach Methoden oder Eigenschaften (E2169)	1141	830
Felddeklarationen sind im Typ interface nicht erlaubt (E2209)	1143	Formular-Designer
Felder	954	763
Klassen und Objekte	954	Formulare
Felder ausrichten (Delphi)	1052	514
Felder deklarieren	647, 1419	FOR-Schleifenvariable '%s' kann nach Durchlauf undefiniert sein (W1037)
Felder hinzufügen	293	1150
Felder-Editor	312	FOR-Schleifenvariable '%s' kann nicht als Var-Parameter übergegeben werden (W1015)
Feld-Offset kann für Variant-Record nicht festgestellt werden, weil der vorherige Feldtyp ein Record-Typ unbekannter Größe ist (E2417)	1252	1150
Feldverbindungs-Designer	311	FOR-Schleifenvariable muss eine einfache lokale Variable sein (x1019)
Fenster Ereignisprotokoll	792	1152
Fenster Liste überwachter Ausdrücke	802	FOR-Schleifenvariable muss von ordinalem Typ sein (E2032)
Fenster Module	798	1151
Fenster Thread-Status	801	forward- und interface-Deklarationen
Fenster: Meldungen	808	Prozeduren und Funktionen
Fensterliste	832	931
Festlegen von C++-Standardprojektoptionen	76	FPU
Filter-Editor	609	794
finalization-Abschnitt	847	Frame verankern
Finalize		1317
Delphi für .NET	1027	Frames verankern
Final-Methoden müssen virtuell oder dynamisch sein (E2351)	1146	185
Flush (Funktion)		Fremdschlüsselbedingung
Gerätetreiber	989	313
for loops	862	FTP-Verbindungsoptionen
FOR oder WHILE Schleife wird nicht durchlaufen - gelöscht (H2135)	1217	382
for...in statements	862	Funktion benötigt Ergebnistyp (E2023)
		1203
		Funktion zur Dokumentationserzeugung konfigurieren
		223
		Funktionen importieren
		Prozeduren und Funktionen
		931
		Funktionsaufrufe
		Ausdrücke
		877
		Funktionsdeklarationen
		Prozeduren und Funktionen
		931
		Für '%s' ist ein Vorgabewert erforderlich (E2238)
		1121
		Für den Zugriff auf '%s' von Unit '%s' wird die Referenz auf importierte Daten (\$G) benötigt (E2201)
		1225
		Für den Zugriff auf Klasseneigenschaften muss ein Klassenfeld oder eine statische Klassenmethode verwendet werden (E2355)
		1109

G

Geerbte Methoden	
Methoden	956
Geerbte Methoden können für den Zugriff auf	

Interface-Eigenschaften nicht verwendet werden (E2370) 1232 Hilfe zur Hilfe 1378
Gemeinsam genutzte Projekte
 aktivieren 258
 in TCC/TAR erstellen 258
 in Together öffnen 258
 Ordnerstruktur erstellen 258
Gemeinsame Nutzung eines Projekts in TCC/TAR und RAD Studio 258
Gemeinsame Nutzung von Pattern 235
Gemeinsame Nutzung von Speicher 101
Generische sortierte Liste 748
Gerätetreiber 989
Gespeicherten Desktop löschen 804
Getter oder Setter für Eigenschaft '%s' kann nicht gefunden werden (E2234) 1190
Globale Prozedur oder statische Klassenmethode erwartet (E2366) 1204

'GOTO %s' führt in oder aus einer TRY-Anweisung (E2127) 1153

G

goto statements 862
Größe 340
Größe des Datentyps ist null (E2101) 1248
Größe von Modellementen ändern 131
Größe von Published-Menge '%s' ist >4 Byte (E2187) 1236
Grundlegende syntaktische Elemente 857
Gruppe überwachter Ausdrücke hinzufügen 782
GUI-Komponenten für die Modellierung 1267
GUI-Komponenten für die Qualitätssicherung 1273
GUI-Komponenten für Pattern 1271

H

Haltepunkt wurde in eine Zeile gesetzt, die weder Quelltext noch Debug-Informationen enthält. 355
HIGH kann nicht auf lange Stringtypen angewendet werden (E2198) 1156
Hilfe
 typographische Konventionen 1378

 hinting directives
 compiler directives,, delphi 862
 Hinweise (Delphi) 1060
 Hinzufügen 373
 Hinzufügen von Teilnehmern zu FCC-Pattern 230
 Historienelement (Zustandsmaschinendiagramm) 1323
 Historienelement erstellen 188
 HPP-Ausgabe (Delphi) 1060
 HR HTML-Element 430
 HTML Tidy-Optionen 751
 HTML/ASP.NET-Optionen 749
 HTML-Formatierung 750
 Hyperlinks in Diagrammen 125

I
IBUpdateSQL- und IBDataSet-Editor (Dialogfeld) 317
IconView-Eintragseditor 619

IDE
 Design-Oberfläche 1363
 Formulare 1363
 MS-Eingabehilfen 1363
 Objektablage 1363
 Objektinspektor 1363
 Projektverwaltung 1363
 Quelltext-Editor 1363
 Tool-Palette 1363
 Willkommensseite 1363
IDE-Befehlszeilenoptionen 1333
if statements 862
 with statements 862
IFDEF-Direktive (Delphi) 1060
IFDEF-Direktive (Delphi) 1061
IFEND-Direktive (Delphi) 1062
IFNDEF-Direktive (Delphi) 1062
IFOPT-Direktive (Delphi) 1063
IMPLEMENTATION-Abschnitt fehlt oder ist falsch 353
Interface 1006
 Im interface-Abschnitt deklarierte Inline-Funktion darf kein

lokales Symbol '%s' verwenden (E2441) 1174	Inkompatible Typen: '%s' (E2009) 1115
Im OLE-Automatisierungsbereich sind nur register-Aufrufkonventionen zulässig (E2179) 1209	Inkompatible Typen: '%s' und '%s' (E2010) 1116
Imagebase \$%X ist kein Vielfaches von 64 K. Wird auf \$%X abgerundet (W1043) 1164	Inline Assembler Stack-Überlauf (E2106) 1093
Imagebase ist zu groß - Programmumfang mehr als 2 GB (E2227) 1164	Inline Assembler Syntaxfehler (E2105) 1093
Image-Basisadresse 1063	Inline für Variable (Dialogfeld) 691
IMG HTML-Element 432	inline-Direktive 945
implementation-Abschnitt 847	Inline-Direktive im Konstruktor oder Destruktor nicht zulässig (E2442) 1173
Implementierung der Interface-Methode %s.%s fehlt (E2291) 1175	Inline-Funktion '%s' wurde nicht expandiert, weil auf Zugriffselement '%s' nicht zugegriffen werden kann (H2444) 1173
Implements-Klausel ist nur für Eigenschaften von Klassen- und Interface-Typen erlaubt (E2259) 1167	Inline-Funktion '%s' wurde nicht expandiert, weil ihre Unit '%s' nicht in der USES-Anweisung des IMPLEMENTATION-Abschnitts angegeben ist und die aktuelle Funktion eine Inline-Funktion ist (H2445) 1173
Implements-Klausel ist nur für lesbare Eigenschaften erlaubt (E2261) 1168	Inline-Funktion '%s' wurde nicht expandiert, weil Unit '%s' in der USES-Liste nicht angegeben ist (H2443) 1173
Implements-Klausel ist nur innerhalb von Klassentypen erlaubt (E2258) 1166	Inline-Funktion darf kein Argument für ein Open Array haben (E2439) 1220
Implements-Klausel kann nicht zusammen mit der Index-Klausel verwendet werden (E2260) 1164	Inline-Funktion darf keinen asm-Block haben (E2426) 1173
Implizite Verwendung der Variants-Unit (W1040) 1169	Inline-Methoden dürfen nicht virtual oder dynamic sein (E2425) 1173
Implizites Erstellen (Delphi) 1064	Inline-Methodensichtbarkeit muss niedriger oder gleich der Sichtbarkeit des Zugriffselements '%s.%s' sein (H2440) 1187
Importieren	Innere Klassifizierer 1298
Nmespaces 1415	erstellen 209
Importierte Daten 1064	Innere Klassifizierer erstellen 209
Importierter Bezeichner '%s' steht im Konflikt mit '%s' in '%s' (x2421) 1111	InOut (Funktion)
Importierter Bezeichner '%s' steht im Konflikt mit '%s' in Namespace '%s' (E2422) 1111	Gerätetreiber 989
In Dateien suchen 675	INPUT HTML-Element 434
In Diagrammen suchen 146	Installieren 283
In Set-Ausdruck WideChar auf Byte-Char verringert (W1050) 1263	Installieren, Starten und Anhalten des externen Debug-Servers 28
In TCC oder TAR erstellte Projekte importieren 252	Installierte .NET-Komponenten 287
In TVS, TEC, TJB oder TPT erstellte Projekte importieren 254	Installierte ActiveX-Komponenten 285
Indexbezeichner	Instanz der abstrakten Klasse '%s' wird konstruiert (E2402) 1117
Eigenschaften 965	Instanz von '%s' mit abstrakter Methode '%s.%s' wird konstruiert (x1020) 1116
Indizes	Instanzelement '%s' in diesem Zusammenhang nicht verfügbar (E2124) 1143
Ausdrücke 877	Instanzspezifikation
Information 596	Funktionsweise definieren 205
Inhalte einfügen (Dialogfeld) 639	instantiiieren 205
initialization-Abschnitt 847	Integer und HResult werden ausgetauscht (x1008) 1157
Inkompatible Typen (E2008) 1113	

Integerkonstante zu lang (E2102) 1177	Typqualifizierer aufzurufen (E2390) 1236
Integer-Typen	Klasse, Interface und Objekttypen sind nur im Abschnitt type erlaubt (E2058) 1219
Datentypen 889	Klassen- und Interface-Typen sind nur in Typabschnitten zulässig (E2060) 1109
Integrierter Assembler 1034	Klassen und Objekte 948
Interaktion 1317	Klassendiagramm
navigieren zu 179	Beispiel 1385
Interaktionsdiagramm	Klassendiagramme als Ansichten verwenden 210
Rückgabebeziehung 158	Klasseneigenschaften
Interaktionsdiagramme	Eigenschaften 965
Navigation 262	Klassenelementdeklarationen nicht zulässig in anonymen Record- oder lokalen Record-Typ (E2435) 1109
Interaktionsverwendung 1317	Klassenfelder
erstellen 179	Felder 954
Interface 1313	Klassenkonstruktor
ausblenden 211	Methoden 956
Notation ändern 214	Klassenkonstruktoren dürfen keine Parameter haben (E2360) 1107
Interface '%s' bereits implementiert von '%s' (E2208) 1177	Klassenkonstruktoren sind in unterstützenden Klassen nicht zulässig (E2358) 1106
Interface '%s' besitzt keine Interface-Identifikation (E2232) 1174	Klassenmethoden 956
Interface '%s' in '%s' ist nicht vollständig definiert (E2420) 1169	Methoden 956
Interface extrahieren/Superklasse extrahieren (Dialogfelder) 688	Klassenmethoden in Record-Typen müssen static sein (E2398) 1238
interface-Abschnitt 847	Klassenreferenzen 974
Interfaces	Klassen und Objekte 974
GUID 1006	Klassentyp erwartet (E2021) 1194
Interface-Typ benötigt (E2205) 1197	Klassentypen
Interne Datenformate 1018	Klassen und Objekte 948
Interne Fehler 1080	Klassenvervollständigung 1373
Interne Fehler beheben 1080	Klassenvervollständigung verwenden 47
Interne Übergänge erstellen 201	Klassifizierer
Interner Fehler: %s%d (F2084) 1174	instantiiieren 205
K	Klassifizierer instantiiieren 150
Keine überladene Version der Array-Eigenschaft '%s' vorhanden, die mit diesen Argumenten verwendet werden kann (E2450) 1214	Knoten
Kiviat-Diagramm	Größe optimieren 131
Kiviat-Diagramm 1456	Größe optimieren (global) 131
Klasse	Kombiniertes Fragment 1320
filtern 194	Kommentar 115
Klasse besitzt bereits eine Standardeigenschaft (E2131) 1124	Kommentar zu Ereignisprotokoll hinzufügen 793
Klasse besitzt keine Standardeigenschaft (E2149) 1195	
Klasse muss sealed sein, um einen private-Konstruktor ohne	

Kommentarblöcke	1373	OLE-Automatisierungsbereich nicht zulässig (E2177)	1117
Kommentare (Delphi)		Kurze String-Typen	
Compiler-Direktiven (Delphi)	857	String-Typen	896
Kompatibilität und Identität von Typen	921	Kurzhinweise während des Debuggens verwenden	23
Komponente	509		
Komponente installieren	284		
Komponenten		L	
Methoden	956	Label '%s' ist in der aktuellen Prozedur nicht deklariert (E2093)	
Komponenten in ein Formular einfügen	55	1211	
Komponenteneigenschaften festlegen	73	Label erwartet (E2031)	1198
Komponenten-Template erzeugen	280	Label ist bereits definiert: '%s' (E2073)	1183
Komponenten-Templates	60	Label wurde deklariert und referenziert, aber nicht gesetzt: '%s' (E2074)	1184
Komponenten-Templates erzeugen	60	Labeldeklaration ist im Interface-Abschnitt nicht erlaubt (E2049)	
Konditionale Blöcke hinzufügen	153	1183	
Konditionaler Block	1304	Labels (Delphi)	857
hinzufügen	153	Laden eines Bildes zur Entwurfszeit	633
Typ festlegen	153	Länge des Ressourcen-Strings überschreitet die Windows-Grenze von 4096 Zeichen (E2381)	1239
Konfigurationsebenen	1280	Lange Strings	
Konfigurations-Manager	6, 7	String-Typen	896
Konfigurieren des Speichermanagers	95	Laufzeitfehler	1079
Konstante 0 wurde zu NIL konvertiert (W1013)	1265	Laufzeit-Package hinzufügen	487
Konstante erwartet (E2109)	1089	Laufzeit-Typinformationen (Delphi)	1073
Konstante oder Typenbezeichner erwartet (E2007)	1119	Layout	
Konstanten		einrichten	127
Datentypen	926	Lebenslinie	
Konstanten dürfen nicht als Argumente für offene Arrays verwendet werden (E2192)	1119	zuordnen	179
Konstantenausdruck erwartet (E2026)	1118	Lebenslinien mit Klassifizierern verknüpfen	175
Konstantenausdruck verletzt untere Grenzen (x1012)	1099	Lebenslinientyp	
Konstantendeklarationen nicht zulässig in anonymen Record- oder lokalen Record-Typ (E2437)	1118	zuordnen	179
Konstantenobjekt kann nicht als Var-Parameter übergegeben werden (E2197)	1119	Lesen einer Nur-Schreiben-Eigenschaft nicht möglich (E2130)	
Konstruktoren		1231	
Delphi für .NET	1027	Linker	515, 568
Methoden	956	Linker Ausgabeoptionen	570
Konstruktoren können mit Instanzvariablen nicht aufgerufen werden (E2382)	1174	Linker Linken	565
Konstruktoren und Destruktoren müssen die %s-Aufrufkonvention haben (E2236)	1209	Linker Warnungen	575
Konstruktoren und Destruktoren sind im		Linker-Fehler bei der Ausgabe des Attributs '%s' (E2327)	1105
		Linker-Fehler bei der Ausgabe von Metadaten (E2328)	1251
		Linker-Fehler: %s	1239
		Linker-Fehler: %s (x1054)	1188
		Linker-Fehler: %s: %s	1240

Liste der Compiler-Direktiven	1052	Mehrzeilen-Editor	825
Liste der Haltepunkte	783	Meldungsfenster	
Listeneditor	577	Refactoring	1417
ListView Items Editor	623	Member	1299
LiveSource-Regeln	1297	Member für einen Zustand erzeugen	189
Logische (bitweise) Operatoren		Member in übergeordnete Klasse verschieben/Member in abgeleitete Klasse verschieben (Dialogfelder)	700
Operatoren	877	Mengen (Sets) dürfen nur maximal 256 Elemente besitzen (E2028)	1241
Lokale Daten zuweisen	294	Mengenkonstruktoren	
Lokale Deklarationen		Ausdrücke	877
Prozeduren und Funktionen	931	Mengenoperatoren	
Lokale Klasse, Interface oder Objekttypen sind nicht erlaubt (E2059)	1218	Operatoren	877
Lokale Klassen- oder Interface-Typen sind nicht zulässig (E2061)	1109	Mengentypen	902
Lokale Prozedur/Funktion '%s' wurde Prozedurenvariable zugewiesen (E2094)	1186	Menü auswählen	643
Lokale Symbolinformationen (Delphi)	1066	Menüs	1267
Lokale Variablen	797	MESSAGE-Direktive (Delphi)	1067
Lokale Variablen können nicht initialisiert werden (E2195)	1171	Metadaten - Datei ist schreibgeschützt (E2331)	1107
Lokaler PInvoke-Quelltext wurde nicht erstellt, weil die externe Routine '%s' im Package '%s' mit package-lokalen Typen in den benutzerdefinierten Attributen definiert wurde (W1053)	1185	Metadaten - Daten zu groß (E2340)	1107
Lokalisierung von Anwendungen – Anleitungen	1394	Metadaten - Daten zu groß (E2344)	1108
LOOP/JCXZ außerhalb des Wertebereichs (E2120)	1183	Metadaten - Datenbank ist schreibgeschützt (E2338)	1108
loops	862	Metadaten - Datenwert wurde abgeschnitten (E2333)	1108
Löschen		Metadaten - Der Import-Gültigkeitsbereich ist mit dem Ausgabe-Gültigkeitsbereich nicht kompatibel (E2339)	1107
Referenzen	1423	Metadaten - Eine erforderliche GUID wurde nicht bereitgestellt (E2346)	1190
M		Metadaten - Erzeugung des gemeinsam genutzten Speichers fehlgeschlagen Eine Speicherzuordnung mit demselben Namen ist bereits vorhanden. (E2336)	1108
Makros		Metadaten - Falsche binäre Signatur (E2347)	1190
aufzeichnen	45	Metadaten - Falsche Eingabeparameter (E2348)	1190
Manager für Build-Konfigurationen	6, 7, 496	Metadaten - Fehler beim Lesen (E2329)	1107
Konfigurations-Manager	4, 14, 15	Metadaten - Fehler beim Öffnen des gemeinsam genutzten Speichers (E2335)	1108
Maskentext-Editor	636	Metadaten - Fehler beim Schreiben (E2330)	1107
MDA	1451	Metadaten - Fehler: alte Version (E2334)	1107
Mehrdimensionale dynamische Arrays		Metadaten - Im Arbeitsspeicher oder Stream befinden sich keine .CLB-Daten (E2337)	1108
Arrays	902	Metadaten - Kein logischer Platz zum Erzeugen weiterer Benutzer-Strings vorhanden (E2350)	1190
Mehrere Elemente erstellen	138	Metadaten - Kein wohlgeformter Name vergeben (E2332)	1108
Mehrere Klassenkonstruktoren in Klasse %s: %s und %s (E2359)	1193	Metadaten - Primärschlüsselpalte lässt möglicherweise keine Nullwerte zu (E2343)	1107
Mehrfachübergänge erstellen	202		

Metadaten - Spalte darf nicht geändert werden (E2341) 1107	navigieren zu 274
Metadaten - typeref kann nicht aufgelöst werden (E2349) 1190	Metriken ausführen 273
Metadaten - Versuch, ein Objekt zu definieren, das bereits vorhanden ist (E2345) 1190	Metrikergebnis (Fenster) 1274
Metadaten - Zu viele RID- oder Primärschlüssel Spalten, max 1 (E2342) 1107	Metrikergebnisse anzeigen 274
Metadaten von .NET-Assemblierungen untersuchen 68	Mindestgröße für Enum-Typen (Delphi) 1068
Methode '%s' mit identischen Parametern und Ergebnistyp ist bereits vorhanden (E2301) 1134	Mit Audit-Sets arbeiten 271
Methode '%s' nicht in Basisklasse gefunden (E2137) 1222	Mit bereitgestellten und erforderlichen Interfaces arbeiten 207
Methode '%s' verbirgt virtuelle Methode vom Basistyp '%s' (W1010) 1154	Mit Beziehungen arbeiten 217
Methode extrahieren 649	Mit der SourceControl arbeiten 1408
Codefragmente 1415	Mit der Versionsverwaltung arbeiten 52
Methode extrahieren (Dialogfeld) 689	Mit einem ausgeführten Prozess verbinden 18
Methoden 956	Mit Feldern arbeiten 216
Klassen und Objekte 956	Mit Instanzspezifikationen arbeiten 205
Methoden des Typs dispinterface dürfen keine Direktiven spezifizieren (E2230) 1130	Mit Interfaces arbeiten 211
Methoden extrahieren	Mit Kollaborationsverwendungen arbeiten 172
Refactoring 1414	Mit kombinierten Fragmenten arbeiten 183
Methoden für Dispatch-Interfaces	Mit komplexen Zuständen arbeiten 199
Automatisierungsobjekte 1015	Mit Konstruktoren arbeiten 215
Methoden überladen	Mit Metrik-Sets arbeiten 276
Methoden 956	Mit Namespaces und Paketen arbeiten 225
Methoden umbenennen 1413	Mit Objektfluss- und Kontrollflussbeziehungen arbeiten 165
Methoden zum Setzen und Lesen von Eigenschaften dürfen nicht überladen werden (E2271) 1221	Mit Prozess verbinden 656
Methoden zum Setzen von Eigenschaften können keine var-Parameter übernehmen (E2282) 1241	Mit referenzierten Projekten arbeiten 266
Methoden-Auflösung ist für Interface '%s' nicht erlaubt (E2264) 1165	Mit UML 1.5-Nachrichten arbeiten 158
Methodenbezeichner erwartet (E2096) 1189	Mit UML 2.0-Nachrichten arbeiten 182
Methodendeklarationen nicht zulässig in anonymen Record- oder lokalen Record-Typ (E2433) 1189	Modell
Metrik 1456	erneut laden 265
Beschreibung anzeigen 274	Modell "analysis"
Metrikdiagramme erstellen 272	Elemente hinzufügen 258
Metriken	Modell "requirements"
Ergebnisse aktualisieren 274	Elemente hinzufügen 258
Ergebnisse sortieren 274	Modellansicht 1270
Metrikergebnisse filtern 274	Modellansicht, Diagrammansicht und Quelltext synchronisieren 262
	Modellelement
	aus- und einblenden 193
	navigieren zu 262
	öffnen 262
	Modellelemente aus- und einblenden 193
	Modellelemente ausrichten 120

Modellelemente auswählen 132
Modellelemente kopieren und einfügen 123
Modellelemente nach Pattern erstellen 227
Modellelemente verschieben 128
Modellunterstützung 694
Modul hinzufügen 781
Modulkopf fehlt oder ist fehlerhaft 352
MSBuild 1461
Multicast-Ereignisse
 Ereignisse 971

N

Nachricht
 Operationsaufruf 182
 Selbst- 182
Nachrichtenbeziehungen einer Methode zuordnen 218
Nachrichtenbeziehungen verzweigen 156
Namen der Zieldatei ändern 381
Namenszuordnung 1452
Namespace
 anzeigen 225
 Löschen 225
 öffnen 225
 umbenennen 225
Namespace hinzufügen 646
Namespace-Konflikte mit Unit-Name '%s' (E2399) 1139
Namespaces 854
 Programme und Units 854
Namespaces in Delphi 854
Namespaces, deklarieren 854
Namespaces, Multi-Unit 854
Namespaces, suchen 854
Native BS-Exceptions 754
Navigation durch Quelltext 1373
Navigation im Diagramm mit der Übersicht 192
Neue Anwendung 392
Neue ASP.NET-Anwendung 387
Neue Dynamische Link-Bibliothek 390
Neue Komponente 290

Neue Konsolenanwendung 388
Neue Projektexperten in Together 1275
Neue SOAP-Server-Anwendung 407
Neue Tags 755
Neue Verbindung 319
Neue Web-Server-Anwendung 411
Neue WebSnap-Anwendung 414
Neuen Web-Service hinzufügen 405
Neuen Web-Service hinzufügen 412
Neuen Wert eingeben 790
Neuer Ausdruck 669
Neuer Kategorienname 485
Neuerungen in RAD Studio (C++Builder) 1358
Neuerungen in RAD Studio (Delphi) 1355
Neues Diagramm hinzufügen (Dialogfeld) 682
Neues Feld 320
Neues Menü anpassen 380
Neues Remote-Datenmodulobjekt 393
Neues Thread-Objekt 394
Neues WebSnap-Datenmodul 416

'Never-build'-Package '%s' muss neu compiliert werden (E2225)
1225

N

New wird für dynamische Arrays nicht unterstützt - es muss Setlength verwendet werden (E2255) 1258
Nicht abgeschlossene bedingte Direktive (E2280) 1254
Nicht abgeschlossener String (E2052) 1254
Nicht aufgelöstes benutzerdefiniertes Attribut: %s (E2289) 1252
Nicht genügend Parameter (E2035) 1192
Nicht verfügbare Optionen 587
Nicht verfügbarer Wert (E2142) 1258
Nicht verwalteter Code 1369
NODEFINE 1070
NOINCLUDE (Delphi) 1070
Notation für Diagramme ändern 117
Nullterminierte Strings
 Strings 989

Numerischer Überlauf (E2113) 1091

Nur eine Methode aus einer Gruppe überladener Methoden darf 'published' sein (E2266) 1136

Nur externe cdecl-Funktionen dürfen varargs verwenden (E2277) 1259

Nur Methoden von abgeleiteten Typen dürfen über Assemblierungsgrenzen hinweg auf das protected-Symbol [%s]%s.%s zugreifen (E2363) 1226

O

Objekt einfügen 631

Objekt oder Klassentyp erforderlich (E2020) 1199

Objektablage 721

Objektdatei linken (Delphi) 1066

Objektdateien linken

Prozeduren und Funktionen 931

Objekten einen Klassifizierer zuordnen 154

Objektgalerie 391

Objekt-Info bearbeiten 716

Objektinspektor 757, 809, 1268

Objekt-Interfaces 1006

Interfaces 1006

Objekttyp erforderlich (E2019) 1199

OCL 1453

Ausdruck 1453

Einschränkungen 1453

unterstützte Diagrammtypen 1453

OCL und ECO-AktionsspracheAusdruckseditor 331

OCL-Ausdrücke bearbeiten 197

OCL-Einschränkungen ein- und ausblenden 198

OCL-Einschränkungen erstellen 196

OCL-Objekteinschränkung

erstellen 196

Offene Array-Parameter

Parameter 938

Offene Arrays 945

Offene String-Parameter (Delphi) 1068

Ö

Öffnen 395

Öffnen (Dialogfeld) 638

Öffnen des C#-Beispielprojekts "Cash Sales" 1381

O

Online-Hilfe 1378

filtern 114

verwenden 114

Open (Funktion)

Gerätetreiber 989

Operanden 1320

Operandengröße stimmt nicht überein (E2107) 1091

Operation

Parameter definieren 174

Operator 1320

Operator ist auf diesen Operandentyp nicht anwendbar (E2015) 1263

Operator oder Semikolon fehlt (E2066) 1191

Operator und Operand für ein kombiniertes Fragment 1320

Operatoren

Ausdrücke 877

Klassen und Objekte 974

Operatoren überladen

Klassen und Objekte 984

Optimierung (Delphi) 1068

Option suchen 564

Optionen 598

Änderungen deaktivieren 102

Optionen des Experten für Datenbindungen 422

Optionen für Distribution über das Web 601

Optionen für Übersetzungs-Tools 765

Optionsgruppe anwenden 492

Optionswert-Editoren 1280

Ordinale Typen

Datentypen 889

Ordinaltyp erforderlich (E2001) 1220

Ordner erstellen 238

Ordner- oder Verzeichnisansicht hinzufügen 524

P

- Package 396
- Package '%s' enthält bereits die Unit '%s' (E2200) 1224
- Package '%s' kann nicht compiliert werden (F2220) 1224
- Package '%s' konnte nicht importiert werden, weil es die System-Unit '%s' enthält (E2416) 1182
- Package '%s' verwendet oder exportiert nicht '%s.%s' (H2235) 1225
- Package '%s' wird benötigt, konnte aber nicht gefunden werden (E2202) 1192
- Package '%s' wird nicht auf Platte geschrieben, da Option –J aktiviert ist (W1031) 1224
- Package %s wird rekursiv benötigt (E2214) 1237
- Package ändern 481
- Package-Import suchen 663
- Packages 289, 519, 764, 1002
- compilieren 3, 1002
 - Units 3
- Packages erstellen 3
- Packages für den EcoSpace auswählen 332
- Package-Unit '%s' kann nicht in den Klauseln contains oder uses erscheinen (E2212) 1255
- PACKED ist hier nicht erlaubt (E2006) 1225
- Packen 252
- Parameter 938
- Prozeduren und Funktionen 938, 945
 - Parameter '%s' hat kein zugehöriges param-Tag im XML-Kommentar für '%s' (aber andere Parameter haben dieses Tag) (W1208) 1264
 - Parameter '%s' ist hier wegen des Vorgabewerts nicht erlaubt (E2237) 1122
 - Parameter ändern 644
 - Refactoring 1422
 - Parameter ändern (Dialogfeld) 684
 - Parameter dieses Typs dürfen keine Standardwerte haben (E2268) 1124
 - Parameter hinzufügen 645
 - Parameter, übergeben 1030
 - Parameterlose Konstruktoren sind in Record-Typen nicht zulässig (E2394) 1226
 - Pattern
- Stub-Implementierung 228
- Pattern als First-Class-Citizens 1454
- Teilnehmer hinzufügen 230
 - Pattern an Verknüpfungen zuweisen 236
 - Pattern exportieren 233
 - Pattern in "Cash Sales" verwenden 1382
 - Pattern sortieren 245
 - Pattern-Experte 1277
 - Pattern-Organizer 1271, 1454
 - Pattern-Organizer öffnen 243
 - Pattern-Registrierung 1272, 1454
 - PE (Portable Executable) Header-Flags (Delphi) 1078
 - Pentium-sichere FDIV-Operationen (Delphi) 1069
 - Pfade und Definitionen 578
 - Pfade und Verzeichnisse (C++) 733
 - Pin
 - erstellen 161 - Pins 1326
 - Pins erstellen 161
 - Ports erstellen 170
 - Positionsmarken
 - Quelltext-Editor 1373
 - Positionsmarken verwenden 46
 - Potentiell polymorphe Konstruktoraufufe müssen virtuell sein (E2391) 1211
 - Pre-Build- oder Post-Build-Ereignis 520
 - PRIVATE oder PROTECTED erwartet (E2375) 1227
 - PROCEDURE oder FUNCTION erwartet (E2122) 1227
 - PROCEDURE, FUNCTION oder CONSTRUCTOR erwartet (E2357) 1227
 - PROCEDURE, FUNCTION, PROPERTY oder VAR erwartet (E2123) 1109
 - Programme und Units 847
 - Programmierhilfe 1373
 - Programmkopf 847
 - Programmorganisation
 - Programme und Units 847
 - Programmstruktur 847
 - Projekt
 - gemeinsam nutzen 258

- importieren 252, 254
Projekt im IBM Rational Rose-Format (MDL) importieren 251
Projekt im XMI-Format importieren 256
Projekt in das XMI-Format exportieren 250
Projektdokumentation erzeugen 224
Projekte 61
 Arten 1369
 zusätzliche Projekte 1369
Projekte aktualisieren 399
Projekteigenschaften 580
Projektoptionen 517
Projektoptionen einstellen 75
Projektreferenzen 476
Projekttypen und -formate für das Modellieren 1332
Projekt-Upgrade 398
Projektverwaltung 810
Protected-Element '%s' ist in diesem Zusammenhang nicht verfügbar (E2389) 1234
Prozedur DISPOSE benötigt einen Destruktor (E2080) 1128
Prozedur FAIL nur im Konstruktor erlaubt (E2078) 1141
Prozedur kann keinen Ergebnistyp besitzen (E2025) 1215
Prozedur NEW benötigt einen Konstruktor (E2079) 1206
Prozedur- oder Funktionsname erforderlich (E2121) 1202
Prozedurale Typen 914
 Datentypen 914
Prozedurdeklarationen
 Prozeduren und Funktionen 931
Prozeduren und Funktionen 930, 931
 Überladen 931
Prozeduren und Funktionen aufrufen 945
 Prozeduren und Funktionen 945
Prozedurendefinition muss der ILCODE-Aufrufkonvention entsprechen (E2297) 1099
Prozess laden Extern 667
Prozess laden Lokal 666
Prozess laden Symboltabellen 668
Prozess laden Umgebungsblock 665
Prüfung von Fließkomma-Exceptions (Delphi) 1059
Prüfung von Var-String (Delphi) 1074
PUBLISHED verursachte, dass RTTI (\$M+) zu Typ '%s'
hinzugefügt wurde (W1055) 1246
Published-Eigenschaft '%s' kann nicht vom Typ %s sein (E2188) 1095
Published-Methode '%s' enthält einen Typ, der nicht als published verwendet werden kann (E2218) 1102
Publizierte Methoden zum Setzen und Lesen von Eigenschaften müssen die %s-Aufrufkonvention besitzen (E2270) 1210
- Q**
- QS-Audits (Dialogfeld) 701
QS-Metriken (Dialogfeld) 703
Quelloptionen 758
Quelltext
 Templates 43, 50
Quelltext bearbeiten
 Code Insight 48
 Code-Snippets 50
 Klassenvervollständigung 47
 Quelltext ausblenden 42
Quelltext bearbeiten – Anleitungen 1393
Quelltext nach Verwendungen durchsuchen 147
Quelltext-Browser 1373
Quelltext-Editor 1373
 anpassen 44
Quelltext-Editor anpassen 44
Quelltexthaltepunkt hinzufügen 839
Quelltexthaltepunkt hinzufügen/Adresshaltepunkt hinzufügen/Datenhaltepunkt hinzufügen 1349
Quelltexthaltepunkte setzen und bearbeiten 19
Quelltext-Hinweise 1373
Quelltext-Templates 1454
 Quelltext-Editor 1373
Quelltext-Templates erstellen 43
Quelltext-Templates verwenden 50
Quelltextvorlagen 731
- R**
- RAD StudioDialog-Hilfe 277
Rangfolge von Operatoren
 Operatoren 877

Raster und andere Optionen für das Erscheinungsbild verwenden	118	Refactoring: Inline-Variablen erstellen	107
read/write für CLR-Ereignisse nicht zulässig. Verwenden Sie eine Include/Exclude-Prozedur (E2298)	1180	Refactoring: Interfaces extrahieren	104
READ/WRITE-Eigenschaftszugriffsmethoden dürfen nicht mit ADD/REMOVE-Zugriffsmethoden gemischt werden (E2404)	1193	Refactoring: Member in die übergeordnete oder abgeleitete Klasse verschieben	111
Real48-Kompatibilität (Delphi)	1071	Refactoring: Member verlagern	110
Record, Objekt oder Klassentyp erforderlich (E2018)	1202	Refactoring: Methoden extrahieren	105
Records (erweiterte)		Refactoring: Parameter ändern	103
Records	902	Refactoring: Sicheres Löschen	112
Records (traditionelle)		Refactoring: Superklasse extrahieren	106
Records	902	Refactoring: Variablen einführen	109
Record-Typ zu groß: 1 MB überschritten (E2419)	1237	Refactoring-Operationen in der Vorschau anzeigen und durchführen	1417
Redeklaration der Eigenschaft im OLE-Automatisierungsbereich nicht erlaubt (E2181)	1233	Refactorings	652
Redeklaration von '%s' verbirgt ein Mitglied (Member) in der Basisklasse (W1009)	1155	Reference	841
Reelle Typen		Referenzen	
Datentypen	889	Projekte	56
Refactoring	1412	Referenzen hinzufügen	56
Anleitungen	1415	Referenzen in lokalen Pfad kopieren	59
durchführen	1417	Referenzen suchen	1423
Feld einführen	108	Deklarationssymbol	1421
In übergeordnete/abgeleitete Klasse verschieben	111	lokale Referenzen suchen	1423
Inline-Variable	107	Referenzierte ECO-Packages	333
Interface extrahieren	104	Referenziertes Projekt	266
Member verlagern	110	Registrieren von Speicherlecks	100
Methode extrahieren	105	Reihenfolge der Felder in der Record-Konstante unterscheidet sich von der Deklaration (E2083)	1236
Parameter ändern	103	Rekursive Include-Datei %s (E2245)	1237
Quelltext-Editor	1373	Relationale Operatoren	
Ressourcen-Strings extrahieren	1417	Operatoren	877
Sicheres Löschen	112	Relozierte Symbole dürfen nicht addiert oder subtrahiert werden (E2111)	1092
Superklasse extrahieren	106	rename symbol	11
Symbol umbenennen	1415	repeat loops	862
Variable einführen	109	Reservierte Wörter (Delphi)	857
Vorschau	1417	Reservierter Adressraum für Ressourcen (Delphi)	1078
Refactoring rückgängig machen (Delphi, C#)	1422	Ressourcen-Compiler	550
Refactoring von Anwendungen	1412	Ressourcen-Compiler Optionen	551
Refactoring von Quelltext	1415	Ressourcen-Compiler Pfade und Definitionen	552
Refactoring: Felder einführen	108	Ressourcen-Datei (Delphi)	1072
		Ressourcendateien im Translation-Manager bearbeiten	8, 88
		Ressourcenmodule aktualisieren	93

Ressourcen-String extrahieren	650	'Self' ist nicht initialisiert. Ein geerbter Konstruktor muss aufgerufen werden (E2304) 1120
Strings konvertieren	1415	'Self' ist nicht initialisiert. Vor dem Aufruf der Vorfahrmethode '%s' muss ein geerbter Konstruktor aufgerufen werden (E2303) 1121
RLINK32.DLL kann nicht geladen werden (E2151)	1193	'Self' ist nicht initialisiert. Vor dem Zugriff auf das Vorfahrtsfeld '%s' muss ein geerbter Konstruktor aufgerufen werden (E2302) 1121
Rolle		'Self' wurde mehr als einmal initialisiert (E2306) 1120
verschiedene Klassifizierer binden	172	
Rückgabewert der Funktion '%s' könnte undefiniert sein (W1035)	1215	
Rückgängig machen		
Refactoring	1422	
RUNONLY-Direktive (Delphi)	1072	
S		
Satellite-Assemblierungsexperte	402	Sequenz- oder Kommunikationsdiagramm aus einer Interaktion erstellen
Schema erzeugen	330	177
Schnittstellen implementieren	1009	Sets
Schnittstellen	1009	1456
Schnittstellenreferenzen	1013	verwenden
Schnittstellen	1013	271, 276
Schreibbare typisierte Konstanten (Delphi)	1077	Sicheres Löschen (Dialogfeld)
Schreibweise von Eigenschaftszugriffsmethode %s.%s sollte %s.%s sein (x2367)	1232	705
Schrift	762	Sichtbarkeit
Schriftarten-Editor	610	Klassen und Objekte
Schwaches Packen	1076	948
Schwerwiegende Fehler	1081	Sichtbarkeit der Eigenschaftszugriffsmethode %s sollte mit der Eigenschaft %s.%s übereinstimmen (H2368) 1233
scope	862	Sichtbarkeitsmodifizierer
Sealed Klasse '%s' kann nicht erweitert werden (E2353)	1141	definieren
Segment/Offset-Paare werden in CodeGear 32-Bit-Pascal nicht unterstützt (E2091)	1240	216
Seite bearbeiten (Dialogfeld)	618	Signatur
Seite hinzufügen (Dialogfeld)	615	521
Seitenoptionen des Anwendungsmoduls/Neues WebSnap-Seitenmodul	413	Skalierung
Selbstübergänge erstellen	203	339
SELECT HTML-Element	437	Slice-Standardfunktion für VAR- und OUT-Argument nicht zulässig (E2454) 1242
Self (Bezeichner)		Slot
Methoden	956	hinzufügen
'Self' ist möglicherweise nicht initialisiert worden (E2305)	1120	zuordnen
		205
		Slot-Stereotyp
		definieren
		205
		Slot-Wert
		festlegen
		205
		SOAP-Datenmodul-Experte
		406
		SortFields-Editor
		323
		Source has been modified. Neu compilieren und binden?
		365
		SourceControl
		Dateien
		1408
		Grundlagen
		1408
		Projekte
		1408
		Repository-Grundlagen
		1408
		Spaltensammlungs-Editor
		295

SPAN HTML-Element 439	Statische Arrays
Speicherbezeichner	Arrays 902
Eigenschaften 965	Statische Instanz- oder Klassenmethode erwartet (E2380) 1197
Speichern unter 403	Statische Klassenmethoden
Speicherreferenz erwartet (E2108) 1090	Methoden 956
Speicherverwaltung 98	Statische Methoden
Delphi für .NET 1027	Methoden 956
Delphi für Win32 1017	Stereotyp
Speicherverwaltung auf der .NET-Plattform 1027	definieren 133
Speicherverwaltung auf der Win32-Plattform 1017	zuweisen 133
Sprachen entfernen 483	Stored Procedure (Dialogfeld) 325
Sprachen hinzufügen 482	Strikte Sichtbarkeit
Sprachen zu einem Projekt hinzufügen 86	Klassen und Objekte 948
Sprach-Exception hinzufügen 723	String-Element kann nicht an var-Parameter übergeben werden (E2354) 1243
Sprach-Exceptions 753	Stringkonstante abgeschnitten damit sie in STRING[%ld] passt (W1014) 1242
Sprach-Feature wird nicht unterstützt: '%s' (x1025) 1253	Stringkonstante zu lang (E2114) 1093
Sprach-Übersicht 842	String-Liste laden 635
SQL-Monitor 324	String-Liste speichern (Dialogfeld) 642
Stack-Frames (Delphi) 1077	String-Listen-Editor 625
Standardaktionsklassen (Dialogfeld) 624	String-Literale können maximal 255 Elemente besitzen (E2056) 1243
Standardeigenschaft muss eine Array-Eigenschaft sein (E2132) 1122	String-Operatoren
Standardfunktion NEW erwartet einen dynamischen Array-Typbezeichner (E2307) 1087	Operatoren 877
Standard-Funktion Slice nur als Argument für offene Arrays erlaubt (E2193) 1242	String-Parameter
Standardfunktion TYPEINFO erwartet einen Typbezeichner (E2133) 1246	Parameter 938
Standardparameter	Strings 857
Parameter 938	String-Typen 896
Standardroutinen	Datentypen 896
System-Unit 989	Strukturansicht 816
Standardroutinen und E/A 988, 989	Strukturfeldbezeichner erwartet (E2119) 1090
Standard-Tastaturvorlage 1335	Strukturierte Typen 902
Standardwerte müssen vom Typ Ordinal, Pointer oder vom Typ small Set sein (E2146) 1125	Datentypen 902
Starker Namensschlüssel kann nicht aus Assemblierung %s extrahiert werden (E2408) 1244	Stub-Implementierung 1454
statements 862	Knoten nach Pattern 228
simple statements 862	verwenden 228
STATIC kann nur für nicht-virtuelle Klassenmethoden verwendet werden (E2376) 1182	Stub-Implementierungs-Pattern verwenden 228
	Such-Bytes eingeben 791
	Suchen 674, 837

- Suchen und Ersetzen 679
- Suchpfad für Symboltabelle hinzufügen 488
- Suchpfade für Assemblierung 286
- Symbol '%s' ist plattformspezifisch (W1002) 1158
- Symbol '%s' ist bibliotheksspezifisch (W1001) 1158
- Symbol '%s' ist experimental (W1003) 1158
- Symbol <symbol> wurde nicht gefunden. 366
- Symbol umbenennen
 - Vorschau 1417
- Symbol umbenennen (C++) 653
- Symbol wählen 600
- Symbolbeschreibung 1373
- Symbole (Delphi)
 - Zeichensätze (Delphi) 857
- Symbole exportieren (Delphi) 1058
- Symbole umbenennen
 - Refactoring 11, 1413
- Symbolleiste Desktop 805
- Symbolleisten
 - anpassen 63
- Symbolleisten anpassen 63, 777
- Sync-Bearbeitungsmodus 54, 1373
 - Quelltext bearbeiten 1422
- Sync-Bearbeitungsmodus (Delphi, C#, C++) 1422
- Sync-Bearbeitungsmodus verwenden 54
- Syntax (Delphi)
 - Zeichensätze (Delphi) 857
- Syntaxfehler in Real-Zahl (E2053) 1096
- System.Runtime.CompilerServices.RunClassConstructor nicht gefunden. Unit-Initialisierungsreihenfolge entspricht nicht der Reihenfolge in der uses-Klausel (W1052) 1250
- Systemmakros 681
- Systemvariable überschreiben/Neue Anwendervariable/Anwendervariable bearbeiten 756
- T**
- Tabelle einfügen 447
- Tabelleneigenschaften 820
- Tabellensammlungs-Editor 298
- TABLE HTML-Element 440
- TableMappings Collection-Editor 326
- Tabulator-Reihenfolge bearbeiten 338
- Targets-Dateien 1471
- Tastaturbelegung 752
- Tastaturkürzel
 - weitere 1278
- Tastatur-Makro aufzeichnen 45
- Tastatur-Makros 45
- Tastaturvorlage BRIEF-Emulation 1340
- Tastaturvorlage Epsilon-Emulation 1342
- Tastaturvorlage IDE - Klassisch 1338
- Tastaturvorlage Visual Basic-Emulation 1345
- Tastaturvorlage Visual Studio-Emulation 1344
- Tastenzuordnungen 1335
- TCC- und TAR-Projekte
 - anzeigen 252
- Teilbereichstypen
 - Datentypen 889
- Template einfügen 630
- Templates (Fenster) 1403
- Templates erstellen
 - Quelltext-Templates 43
- Templates in die Objektablage einfügen 58
- Testen von Units – Anleitungen 1402
- Testfall
 - Tests schreiben 1427
- Testfall-Experte 450, 451
- Testprojekt
 - Quelltext zu einem Projekt hinzufügen 1427
- Testprojekt-Experte 452, 453
- Text hinter dem abschließenden 'END.' wird vom Compiler ignoriert (W1011) 1152
- TEXTAREA HTML-Element 442
- Textdateityp
 - Dateitypen 989
- Themes 65
- Thread-lokale Variablen können nicht ABSOLUTE sein (E2190) 1084
- Thread-lokale Variablen können nicht initialisiert werden (E2194) 1172

Thread-lokale Variablen können nicht lokal zu einer Funktion sein (E2189) 1187	Tool-Palette 760, 821, 1267
TObject und TClass	Komponenten 55
Klassen und Objekte 948	Tool-Palette anpassen 64
To-Do-Eintrag hinzufügen oder bearbeiten 818	Tools-Eigenschaften 771
To-Do-Liste 817	Tools-Optionen 715
To-Do-Liste filtern 819	Transaktionaler Objektexperte 408
To-Do-Listen	Transaktions-Editor (Dialogfeld) 316
Planung 82	Translation-Manager 823
Übersicht 1373	TreeView-Eintragseditor 626
To-Do-Listen verwenden 82	Treibereinstellungen 310
Together – Allgemeine Optionen 1281	try... finally-Anweisungen
Together – Anleitungen 113	Exceptions 977
Together – Refactoring-Operationen 1331	try...except-Anweisungen
Together konfigurieren 102	Exceptions 977
Together Object Constraint Language (OCL) – Anleitungen 1397	Turbo Assembler 581
Together- Optionen für Sequenzdiagramm-Roundtrips 680	Turbo Assembler Optionen 582
Together-Diagramme – Anleitungen 1395	Turbo Assembler Pfade und Definitionen 585
Together-Diagrammoptionen (Ansichtsverwaltung) 1289	Turbo Assembler Warnungen 586
Together-Diagrammoptionen (Drucken) 1287	Typ '%s' benötigt Finalisierung - im Dateityp nicht erlaubt (E2155) 1146
Together-Diagrammoptionen (Erscheinungsbild) 1282	Typ '%s' benötigt Finalisierung - nicht im Variant-Record erlaubt (E2154) 1146
Together-Diagrammoptionen (Layout) 1284	Typ '%s' benötigt Initialisierung - im Variant-Record nicht erlaubt (E2418) 1171
Together-Dokumentationserzeugung – Anleitungen 1396	Typ '%s' besitzt keine Typinformation (E2134) 1217
Together-Experten 1275	Typ '%s' ist nicht vollständig definiert (E2086) 1170
Together-Glossar 1266	Typ erwartet (E2110) 1094
Together-Konfigurationsoptionen 1280	Typ in einem OLE-Automatisierungsauftrag nicht erlaubt (E2160) 1095
Together-Optionen (Dialogfeld) 696	Typ ist im Aufruf von Variant Dispatch nicht zulässig (E2281) 1097
Together-Optionen für die Dokumentationserzeugung 1291	Typbibliothek (C++) 734
Together-Optionen für die Modellansicht 1293	Typbibliothek (Delphi) 768
Together-Optionskategorien 1281	Typbibliothek-Explorer 835
Together-Pattern – Anleitungen 1398	Typbibliothekseditor 826
Together-Projekte – Anleitungen 1399	Typdeklaration 923
Together-Projekte – Überblick 1446	Datentypen 923
Together-Qualitätssicherung – Anleitungen 1400	Typdeklarationen sind im anonymen Record- oder lokalem Record-Typ nicht zulässig (E2436) 1247
Together-Quelltextoptionen 1294	TYPEOF kann nur auf Objekttypen mit VMT angewendet werden (E2082) 1204
Together-Refactoring – Anleitungen 1401	Typisierte Konstante '%s' als Var-Parameter übergegeben
Together-Referenz 1266	
Together-Tastaturkürzel 1278	
Together-Unterstützung für Projekte aktivieren 248	
Tool-Fenster andocken 66	

(W1016) 1246	Überblick zur Dokumentationserzeugung 1458
Typkompatibilität	Überblick zur Interoperabilität 1460
Datentypen 921	Überblick zur Modellierung 1446
Typprüfung bei Zeigern (Delphi) 1073	Überblick zur OCL-Unterstützung 1453
Typumwandlung 671	Überblick zur Umwandlung in Quelltext 1452
Typumwandlungen	Übergang 1307
Ausdrücke 877	Selbst- 203
Ü	Übergang oder Zustand mit einer Aktivität verknüpfen 186
Übergänge	mehrere 1307
Überblick über benannte Optionsgruppen 1469	Überkreuzender Bezug zweier Units auf '%s' (F2047) 1108
Überblick über Build-Konfigurationen (C++) 1465	Überladene Operatoren 984
Überblick über Build-Konfigurationen (Delphi) 1463	Überlauf bei Konvertierung oder arithmetischer Operation (E2099) 1087
Überblick über das Testen von Units 1425	Überlaufprüfung (Delphi) 1069
Überblick über DUnit 1429	Überschriebene automatisierte virtuelle Methode '%s' kann kein dispid definieren (E2185) 1160
Überblick über LiveSource 1451	Übersetzungs-Tools
Überblick über NUnit 1432	aktive Sprache festlegen 90
Überblick über Pattern 1454	externen Translation-Manager einrichten 91
Überblick über Qualitätssicherungsfunktionen 1456	externer Translation-Manager 94
Überblick über Together-Diagramme 1447	Ressourcenmodule aktualisieren 93
Überblick zu Hinweisen in Diagrammen 1448	Sprachen zu einem Projekt hinzufügen 86
Überblick zu Hyperlinks 1450	Translation-Manager verwenden 88
Überblick zu MSBuild 1461	Überblick 1437
Überblick zu Namespaces und Paketen 1446	Übersetzungswörterbuch 767
Überblick zu Verknüpfungen 1449	Übersicht über Delphi 842
Überblick zum Ändern von Parametern (Delphi) 1422	Übersicht zu Modellelementen 1448
Überblick zum Debuggen 1439	Übersicht zum Debuggen externer Anwendungen 32
Überblick zum Deklarieren von Variablen und Feldern (Delphi) 1419	Überwachen der Speicherverwendung 99
Überblick zum Diagramm-Layout 1450	Überwachungsbedingung für einen Übergang erstellen 187
Überblick zum externen Debugger 1441	
Überblick zum Extrahieren von Methoden (Delphi) 1414	
Überblick zum Extrahieren von Ressourcen-Strings (Delphi) 1417	
Überblick zum Importieren und Exportieren 1459	U
Überblick zum Refactoring 1412, 1456	UCS-4-Textcodierung nicht unterstützt. Konvertieren Sie in UCS-2 oder UTF-8 (F2438) 1183
Überblick zum Steuern des Entwicklungszyklus 1406	UDDI-Browser 478
Überblick zum Suchen von Referenzen 676	Umbenennen (C#) 654
Überblick zum Suchen von Referenzen (Delphi, C#, C++) 1421	Umbenennen (Delphi) 655
Überblick zum Umbenennen von Symbolen (Delphi, C#, C++) 1413	Umgebungsoptionen 743
	Umgebungsvariablen 744

UML 1447	Unbekannten Elementtyp beim Importieren der Signatur von '%s.%s' gefunden (E2405) 1252
UML 1.5 Together-Experte für Design-Projekte 1275	Unbekanntes Ressourcenformat '%s' (E2400) 1251
UML 1.5-Aktivitätsdiagramm 1308	Unbenannte Argumente müssen benannten Argumenten in der OLE-Automatisierung vorangestellt werden (E2166) 1252
UML 1.5-Aktivitätsdiagramme erstellen 149	UNDEF-Direktive (Delphi) 1074
UML 1.5-Anwendungsfalldiagramm 1300	Undefinierter Bezeichner: '%s' (E2003) 1158
UML 1.5-Interaktionsdiagramm 1302	Unerkannte Schlüsseldatei '%s' für starke Namen (E2388) 1181
UML 1.5-Klassendiagramme 1294	Unerwartetes Dateiende im in Zeile %ld beginnenden Kommentar (E2057) 1254
UML 1.5-Komponentendiagramm 1310	Ungenügend Speicher zum Starten 354
UML 1.5-Komponentendiagramme erstellen 151	Ungenügende Forward- oder External-Deklaration: '%s' (E2065) 1147
UML 1.5-Modellierung mit "Cash Sales" 1381	Ungültige Botschaftsparameterliste (E2138) 1180
UML 1.5-Nachricht 1305	Ungültige Compiler-Direktive: '%s' (x1030) 1179
UML 1.5-Referenz 1294	Ungültige Kombination von Opcode und Operanden (E2116) 1088
UML 1.5-Verteilungsdiagramm 1311	Ungültige Operatordeklaration (E2393) 1181
UML 1.5-Verteilungsdiagramme erstellen 152	Ungültige Referenz auf Symbol '%s' in Objektdatei '%s' (E2068) 1161
UML 1.5-Zustandsdiagramm 1305	Ungültige Registerkombination (E2112) 1092
UML 1.5-Zustandsdiagramme erstellen 160	Ungültige Typumwandlung (E2089) 1178
UML 2.0 Together-Experte für Design-Projekte 1276	Ungültiger Botschaftsmethoden-Index (E2139) 1161
UML 2.0-Aktivitätsdiagramm 1323	Ungültiger Funktionsrückgabetyp (E2024) 1239
UML 2.0-Aktivitätsdiagramme erstellen 162	Ungültiger Typ im OLE-Automatisierungsbereich: '%s' (E2176) 1159
UML 2.0-Anwendungsfalldiagramm 1314	Ungültiger Typ in Read/Readln-Anweisung (E2055) 1096
UML 2.0-Beispielprojekt 1384	Ungültiger Typ in Write/Writeln-Anweisung (E2054) 1098
UML 2.0-Beispielprojekt, Behavior-Paket 1386	Ungültiger Versions-String '%s' in %s angegeben (E2386) 1183
UML 2.0-Beispielprojekt, Structure-Paket 1385	Ungültiges Ereignisprofil <name> 369
UML 2.0-Interaktionsdiagramm 1315	Ungültiges Zeichen in Eingabedatei: '%s' (%s) (E2038) 1162
UML 2.0-Klassendiagramme 1313	Unicode-Zeichen sind in published Symbolen nicht zulässig (E2452) 1249
UML 2.0-Komponentendiagramm 1326	Unit 428
UML 2.0-Komponentendiagramme erstellen 167	Unit '%s' in Package %s verweist auf Unit %s, die in keinem Package vorhanden ist. Package-Unit dürfen nur auf Package-Units verweisen (E2411) 1209
UML 2.0-Kompositionssstrukturdiagramm 1329	Unit '%s' ist bibliotheksspezifisch (W1004) 1250
UML 2.0-Nachricht 1318	Unit '%s' ist experimental (W1007) 1250
UML 2.0-Referenz 1313	Unit '%s' ist plattformspezifisch (W1005) 1251
UML 2.0-Sequenzdiagramme oder -Kommunikationsdiagramme erstellen 179	Unit '%s' veraltet oder beschädigt: '%s' fehlt (E2158) 1097
UML 2.0-Verteilungsdiagramm 1328	Unit '%s' veraltet oder beschädigt: '%s.%s' fehlt (E2159) 1097
UML 2.0-Verteilungsdiagramme erstellen 174	Unit '%s' wird abgelehnt (W1006) 1250
UML 2.0-Zustandsmaschinendiagramm 1321	
UML 2.0-Zustandsmaschinendiagramme erstellen 191	
UML in Farbe 1447	
verwenden 119	
Umwandlung in Quelltext 1452	
Unbekannte Direktive: '%s' (E2070) 1251	

Unit '%s' wird mit Unit '%s' in '%s' kompiliert, aber abweichende Version von '%s' gefunden (F2446) 1126	Untersuchen 664
Unit %s wurde mit einer unterschiedlichen Version von %s.%s kompiliert (F2051) 1261	Untypisierte Dateien Dateitypen 989
Unit nicht gefunden: '%s' oder binäre Äquivalente (%s) (F1027) 1145	User Control einfügen 444
Unit suchen 651	uses-Klausel 847 Unit-Referenzen 847
Unit System nicht kompatibel mit der Testversion (F2087) 1126	
Unit verwenden 410	
Unit-Bezeichner '%s' stimmt mit dem Dateinamen nicht überein (E1038) 1250	Variable '%s' ist möglicherweise nicht initialisiert worden (W1036) 1255
Unit-Namen 854	Variable '%s' wurde deklariert, aber in '%s' nicht verwendet (H2164) 1254
Unit-Namen stimmen nicht überein: '%s' '%s' (E2085) 1193	Variable deklarieren Refactoring 1415
Units	Variable einführen (Dialogfeld) 693
Delphi für .NET 1027	Variable erforderlich (E2036) 1204
linken 10	Variable erwartet (E2088) 1259
Namespaces 1424	Variable und Felder deklarieren, Beispiele 1419
Units anzeigen 831	Variablen 924 Datentypen 924
Units suchen und Namespaces verwenden (Delphi, C#) 1424	Variablen deklarieren 648 Anfangstyp 1419
Units testen 1425, 1427	Variablen deklarieren, Regeln 1419
Unit-Struktur 847	Variablen können nur einzeln initialisiert werden (E2196) 1172
Unsafe-Code: '%s' (W1047) 1253	Variablen und Felder deklarieren 1419
Unsichere Prozeduren sind nur bei der Compilierung mit {\$UNSAFECODE ON} zulässig (E2395) 1253	Variablenausdrücke bearbeiten 26
Unsichere Zeiger sind nur bei der Compilierung mit {\$UNSAFECODE ON} zulässig (E2397) 1253	Variante Teile in Record-Typen 902
Unsichere Zeigervariablen, -parameter oder -konstanten sind nur in unsicheren Prozeduren zulässig (E2410) 1253	Variante Typen 917 Datentypen 917
Unsicherer Code (Delphi für .NET) 1074	VCL für .NET-Quelltext debuggen 22
Unsicherer Code ist nur in unsicheren Prozeduren zulässig (E2396) 1252	VCL-Designer 736
Unsicherer Typ: '%s%s%s' (W1046) 1253	Verbindung für das externe Debuggen einrichten 33
Unterbrechung durch Benutzer - Compilierung abgebrochen (E2090) 1255	Verbindung umbenennen 322
Unterer Bereich überschreitet oberen Bereich (E2011) 1188	Verbindungseditor 302, 303
Unterstützende Klassen 987	Vererbung und Gültigkeitsbereich Klassen und Objekte 948
Klassen und Objekte 987	Vergrößern des Speicheradressraums 97
Unterstützender Klassentyp erforderlich (E2022) 1197	Verknüpfung erstellen 139
Unterstützte C# Projektexperten 1277	Verknüpfung Klassifizierer
Unterstützte Delphi-Projektexperten 1277	
Unterstützte UML-Spezifikationen 1447	
Unterstützung aktivieren 248	

- erstellen 154
- Verknüpfung zu einem Pattern erstellen 239
- Verknüpfung zu einer Interaktion in ein Interaktionsdiagramm einfügen 181
- Verknüpfungen erstellen 139
- Verknüpfungen, Ordner und Pattern-Hierarchien kopieren und einfügen 237
- Verknüpfungen, Ordner und Pattern-Hierarchien löschen 241
- Verlagern (Dialogfeld) 695
- Verschachtelte Exceptions**
 - Exceptions 977
- Verschachtelte Inline-Routine '%s' kann nicht auf Variable '%s' außerhalb des Gültigkeitsbereichs zugreifen (E2449) 1173
- Verschachtelte Konstanten**
 - Verschachtelte Typdeklarationen 982
- Verschachtelte Nachricht 1305
- Verschachtelte Typdeklarationen** 982
- Verschachtelte Typen**
 - Klassen und Objekte 982
- Versionsinformationen** 523
- Versionsverwaltung** 52, 719
- Verteilungsdiagramm**
 - Beispiel 1385, 1386
- Verwenden der CPU-Ansicht 38
- Verwenden des Datei-Browsers 81
- Verwenden von Targets-Dateien 14
- Verwenden von virtuellen Ordnern 83
- Verwendete Unit '%s' kann nicht kompiliert werden (F2063) 1255
- Verwendung des integrierten Assemblers (nur Win32) 1034
- Verwendung suchen (Dialogfeld) 707
- Verwendung von Exceptions**
 - Exceptions 977
- Verzeichnis auswählen 404
- Verzeichnisse/Bedingungen 512
- Verzögertes Ereignis 1308
- Verzögertes Ereignis erstellen 200
- Virtuelle Konstruktoren sind nicht erlaubt (E2062) 1219
- Virtuelle Methoden sind in Record-Typen nicht zulässig (E2379) 1261
- Virtuelle Ordner 83
- Virtuelle Pattern-Hierarchie erstellen 240
- Virtuelle und dynamische Methoden**
 - methods 956
- Virtuelles Verzeichnis konfigurieren 494
- Vista-Themes 1363
- Visual Studio-Projekt exportieren 718
- Void-Typ in diesem Kontext nicht verwendbar (E2423) 1261
- Von RAD Studio erzeugte Dateien 384
- Voreinstellungen für Tools festlegen 79
- Vorhandene Pattern importieren 234
- Vorhandenes Projekt für die Modellierung öffnen 257
- Vorlagen löschen 608
- Vorwärtsdeklarationen**
 - Klassen und Objekte 948
- Vorzeichenbehaftete und -lose Typen werden kombiniert - beide Operanden werden erweitert (W1024) 1111
- Vorzeichenbehaftete und -lose Typen werden verglichen - beide Operanden werden erweitert (W1023) 1112

W

- W1000 Symbol '%s' wird abgelehnt (W1000) 1157
- Warnmeldungen (Delphi) 1075
- Warnungen (Delphi) 1076
- Was ist RAD Studio? 1354
- Web-Anwendungskomponenten 417
- WebSnap 769
- Weitere Programmiersprachen installieren 9
- Wert '%s' für Option %s wurde abgeschnitten (W1049) 1220
- Werte von Datenelementen untersuchen und ändern 24
- Wertlisten-Editor 627
- while loops 862
- WideString
 - String-Typen 896
- WideStrings 989
- Wiederhervorrufen einer Exception ist nur im Exception-Handler möglich (E2145) 1238
- Windows Forms-Designer 770
- Windows-Typbibliotheken untersuchen 69
- with statements 862
- Wörterbuch 766

WSDL: Optionen für den Import 418

X

XMI-Export (Dialogfeld) 710

XMI-Import (Dialogfeld) 711

XML-Kommentar bei '%s' hat das cref-Attribut '%s', das nicht aufgelöst werden konnte (W1206) 1263

XML-Kommentar bei '%s' hat ein param-Tag für '%s', aber es gibt keinen Parameter mit diesem Namen (W1207) 1264

XML-Kommentar in '%s' ist falsch strukturiert -- 'Das Zeichen '%c' wurde erwartet.' (W1205) 1264

XML-Kommentar in '%s' ist falsch strukturiert -- 'Ein Name beginnt mit einem ungültigen Zeichen.' (W1203) 1264

XML-Kommentar in '%s' ist falsch strukturiert -- 'Ein Name enthält ein ungültiges Zeichen.' (W1204) 1264

XML-Kommentar in '%s' ist falsch strukturiert -- 'Referenz auf nicht definierte Entität '%s'.' (W1202) 1264

XML-Kommentar in '%s' ist falsch strukturiert -- 'Whitespace ist an dieser Position nicht zulässig.' (W1201) 1265

XML-Mapper 772

Z

Zahlen (Delphi)

Zeichensätze (Delphi) 857

Zeichensätze (Delphi)

Delphi-Zeichensatz 857

reservierte Wörter 857

Zeichentypen

Datentypen 889

Zeiger und Zeigertypen 911

Zeiger-Operatoren

Operatoren 877

Zeigertyp erforderlich (E2017) 1200

Zeigertypen

Datentypen 911

Zeile zu lang (mehr als 1023 Zeichen) (F2069) 1184

Ziel kann nicht zugewiesen werden (E2453) 1126

Zielprojekt auswählen/Quellprojekt auswählen (Dialogfeld) 685

Zirkuläre Unit-Referenzen

uses-Klausel 847

Zu viele bedingte Symbole (E2163) 1245

Zu viele lokale Konstanten. Verwenden Sie kürzere Prozeduren (E2283) 1245

Zu viele Parameter (E2034) 1245

Zu viele verschachtelte bedingte Symbole (E2279) 1206

Zu wenig Arbeitsspeicher (F2046) 1188

Zu Zeilennummer gehen 678

Zugriff auf Eigenschaft muss über Instanzenfeld oder -methode erfolgen (E2356) 1234

Zugriffsmethode zum Hinzufügen oder Entfernen für Ereignis '%s' nicht gefunden (E2403) 1191

Zustand 1307

verschachtelt 199

Zustände erstellen 190

Zustandsinvariante 1317

verbinden 178

Zustandsinvarianten erstellen 178

Zustandsmaschinendiagramm

Beispiel 1386

Zuweisung an FOR-Schleifenvariable '%s' (E2081) 1088

Zuweisung für typisierte Konstante '%s' (W1017) 1088

Zweifelhafte Typumwandlung von %s in %s (W1044) 1244

Zweifelhafte Typumwandlung von '%s' in '%s' (W1048) 1253

Zwischen UML 1.5-Sequenz- und -Kollaborationsdiagrammen umschalten 157