

holger SCHWICHTENBERG



Effizienter Datenzugriff mit

ENTITY FRAMEWORK CORE

Datenbankprogrammierung mit C#
für .NET Framework, .NET Core und Xamarin

HANSER

Schwichtenberg

Effizienter Datenzugriff mit Entity Framework Core

Bleiben Sie auf dem Laufenden!



Unser **Computerbuch-Newsletter** informiert Sie monatlich über neue Bücher und Termine. Profitieren Sie auch von Gewinnspielen und exklusiven Leseproben. Gleich anmelden unter



www.hanser-fachbuch.de/newsletter



Hanser Update ist der IT-Blog des Hanser Verlags mit Beiträgen und Praxistipps von unseren Autoren rund um die Themen Online Marketing, Webentwicklung, Programmierung, Softwareentwicklung sowie IT- und Projektmanagement. Lesen Sie mit und abonnieren Sie unsere News unter



www.hanser-fachbuch.de/update



Holger Schwichtenberg

Effizienter Datenzugriff mit Entity Framework Core

Datenbankprogrammierung mit C#
für .NET Framework, .NET Core und
Xamarin

HANSER

Der Autor:

Dr. Holger Schwichtenberg, Essen

www.IT-Visions.de

Alle in diesem Buch enthaltenen Informationen, Verfahren und Darstellungen wurden nach bestem Wissen zusammengestellt und mit Sorgfalt getestet. Dennoch sind Fehler nicht ganz auszuschließen. Aus diesem Grund sind die im vorliegenden Buch enthaltenen Informationen mit keiner Verpflichtung oder Garantie irgendeiner Art verbunden. Autor und Verlag übernehmen infolgedessen keine juristische Verantwortung und werden keine daraus folgende oder sonstige Haftung übernehmen, die auf irgendeine Art aus der Benutzung dieser Informationen – oder Teilen davon – entsteht.

Ebenso übernehmen Autor und Verlag keine Gewähr dafür, dass beschriebene Verfahren usw. frei von Schutzrechten Dritter sind. Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Buch berechtigt deshalb auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären und daher von jedermann benutzt werden dürften.

Bibliografische Information der Deutschen Nationalbibliothek:

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.d-nb.de> abrufbar.

Dieses Werk ist urheberrechtlich geschützt.

Alle Rechte, auch die der Übersetzung, des Nachdruckes und der Vervielfältigung des Buches, oder Teilen daraus, vorbehalten. Kein Teil des Werkes darf ohne schriftliche Genehmigung des Verlages in irgendeiner Form (Fotokopie, Mikrofilm oder ein anderes Verfahren) – auch nicht für Zwecke der Unterrichtsgestaltung – reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.

© 2018 Carl Hanser Verlag München, www.hanser-fachbuch.de

Lektorat: Sylvia Hasselbach

Copy editing: Matthias Bloch, Essen

Umschlagdesign: Marc Müller-Bremer, München, www.rebranding.de

Umschlagrealisation: Stephan Rönigk

Gesamtherstellung: Kösel, Krugzell

Ausstattung patentrechtlich geschützt. Kösel FD 351, Patent-Nr. 0748702

Printed in Germany

Print-ISBN: 978-3-446-44898-8

E-Book-ISBN: 978-3-446-44978-7

Inhalt

Vorwort	XV
Über den Autor	XVII
1 Einleitung	1
1.1 Programmiersprache in diesem Buch	1
1.2 Fallbeispiele in diesem Buch	1
1.2.1 Entitäten	2
1.3 Anwendungsarten in diesem Buch	5
1.4 Hilfsroutinen zur Konsolenausgabe	6
1.5 Programmcodebeispiel zum Download	10
2 Was ist Entity Framework Core?	13
2.1 Was ist ein Objekt-Relationaler Mapper?	13
2.2 ORM in der .NET-Welt	15
2.3 Versionsgeschichte von Entity Framework Core	16
2.4 Unterstützte Betriebssysteme	17
2.5 Unterstützte .NET-Versionen	17
2.6 Unterstützte Visual Studio-Versionen	18
2.7 Unterstützte Datenbanken	19
2.8 Funktionsumfang von Entity Framework Core	20
2.9 Funktionen, die dauerhaft entfallen	21
2.10 Funktionen, die Microsoft bald nachrüsten will	21
2.11 Hohe Priorität, aber nicht kritisch	22
2.12 Neue Funktionen in Entity Framework Core	23
2.13 Einsatzszenarien	24
3 Installation von Entity Framework Core	27
3.1 Nuget-Pakete	27

3.2	Paketinstallation	30
3.3	Aktualisierung auf eine neue Version	34
4	Konzepte von Entity Framework Core	39
4.1	Vorgehensmodelle bei Entity Framework Core	39
4.2	Artefakte bei Entity Framework Core	41
5	Reverse Engineering bestehender Datenbanken	43
5.1	Reverse Engineering mit PowerShell-Befehlen	44
5.2	Codegenerierung	47
5.3	.NET Core-Tool	54
5.4	Schwächen des Reverse Engineering	56
6	Forward Engineering für neue Datenbanken	57
6.1	Regeln für die selbsterstellten Entitätsklassen	59
6.1.1	Properties	59
6.1.2	Datentypen	59
6.1.3	Beziehungen (Master-Detail)	59
6.1.4	Vererbung	61
6.1.5	Primärschlüssel	61
6.1.6	Beispiele	61
6.2	Regeln für die selbsterstellte Kontextklasse	64
6.2.1	Nuget-Pakete	64
6.2.2	Basisklasse	65
6.2.3	Konstruktor	65
6.2.4	Beispiel	65
6.2.5	Provider und Verbindungszeichenfolge	66
6.2.6	Eigene Verbindungen	67
6.2.7	Thread-Sicherheit	67
6.3	Regeln für die Datenbankschemagenerierung	67
6.4	Beispiel-Client	68
6.5	Anpassung per Fluent-API (OnModelCreating())	69
6.6	Das erzeugte Datenmodell	71
7	Anpassung des Datenbankschemas	73
7.1	Persistente versus transiente Klassen	74
7.2	Namen im Datenbankschema	75
7.3	Reihenfolge der Spalten in einer Tabelle	75
7.4	Spaltentypen/Datentypen	76
7.5	Pflichtfelder und optionale Felder	77
7.6	Feldlängen	77
7.7	Primärschlüssel	78

7.8	Beziehungen und Fremdschlüssel	78
7.9	Optionale Beziehungen und Pflichtbeziehungen	79
7.10	Uni- und Bidirektionale Beziehungen	81
7.11	1:1-Beziehungen	82
7.12	Indexe festlegen	83
7.13	Weitere Syntaxoptionen für das Fluent-API	85
7.13.1	Sequentielle Konfiguration	85
7.13.2	Strukturierung durch Statement Lambdas	85
7.13.3	Strukturierung durch Unterrouninen	86
7.13.4	Strukturierung durch Konfigurationsklassen	87
7.14	Massenkonfiguration mit dem Fluent-API	88
8	Datenbankschemamigrationen	91
8.1	Anlegen der Datenbank zur Laufzeit	91
8.2	Schemamigrationen zur Entwicklungszeit	92
8.3	Befehle für die Schemamigrationen	92
8.4	ef.exe	93
8.5	Add-Migration	94
8.6	Update-Database	98
8.7	Script-Migration	99
8.8	Weitere Migrationsschritte	99
8.9	Migrationsszenarien	100
8.10	Weitere Möglichkeiten	101
8.11	Schemamigrationen zur Laufzeit	103
9	Daten lesen mit LINQ	105
9.1	Kontextklasse	105
9.2	LINQ-Abfragen	106
9.3	Schrittweises Zusammensetzung von LINQ-Abfragen	108
9.4	Repository-Pattern	109
9.5	Einsatz von var	110
9.6	LINQ-Abfragen mit Paging	111
9.7	Projektionen	113
9.8	Abfrage nach Einzelobjekten	114
9.9	Laden anhand des Primärschlüssels mit Find()	115
9.10	LINQ im RAM statt in der Datenbank	115
9.11	Umgehung für das GroupBy-Problem	119
9.11.1	Mapping auf Nicht-Entitätstypen	119
9.11.2	Entitätsklasse für die Datenbanksicht anlegen	120
9.11.3	Einbinden der Entitätsklasse in die Kontextklasse	120
9.11.4	Verwendung der Pseudo-Entitätsklasse	121

9.11.5	Herausforderung: Migrationen	121
9.11.6	Gruppierungen mit Datenbanksichten	123
9.12	Kurzübersicht über die LINQ-Syntax	123
9.12.1	Einfache SELECT-Befehle (Alle Datensätze)	125
9.12.2	Bedingungen (where)	125
9.12.3	Bedingungen mit Mengen (in)	126
9.12.4	Sortierungen (orderby)	126
9.12.5	Paging (Skip() und Take())	127
9.12.6	Projektion	127
9.12.7	Aggregatfunktionen (Count(), Min(), Max(), Average(), Sum()) ..	128
9.12.8	Gruppierungen (GroupBy)	129
9.12.9	Einzelobjekte (SingleOrDefault(), FirstOrDefault())	130
9.12.10	Verbundene Objekte (Include())	131
9.12.11	Inner Join (Join)	132
9.12.12	Cross Join (Kartesisches Produkt)	133
9.12.13	Join mit Gruppierung	133
9.12.14	Unter-Abfragen (Sub-Select)	134
10	Objektbeziehungen und Ladestrategien	137
10.1	Standardverhalten	137
10.2	Kein Lazy Loading	138
10.3	Eager Loading	140
10.4	Explizites Nachladen (Explicit Loading)	143
10.5	Preloading mit Relationship Fixup	145
10.6	Details zum Relationship Fixup	150
11	Einfügen, Löschen und Ändern	151
11.1	Speichern mit SaveChanges()	151
11.2	Änderungsverfolgung auch für Unterobjekte	154
11.3	Das Foreach-Problem	155
11.4	Objekte hinzufügen mit Add()	156
11.5	Verbundene Objekte anlegen	158
11.6	Verbundene Objekte ändern/Relationship Fixup	161
11.7	Widersprüchliche Beziehungen	164
11.8	Zusammenfassen von Befehlen (Batching)	169
11.9	Objekte löschen mit Remove()	170
11.10	Löschen mit einem Attrappen-Objekt	171
11.11	Massenlöschen	172
11.12	Datenbanktransaktionen	173
11.13	Change Tracker abfragen	176
11.13.1	Zustand eines Objekts	176
11.13.2	Liste aller geänderten Objekte	178

12	Datenänderungskonflikte (Concurrency)	181
12.1	Rückblick	181
12.2	Im Standard keine Konflikterkennung	182
12.3	Optimistisches Sperren/Konflikterkennung	184
12.4	Konflikterkennung für alle Eigenschaften	185
12.5	Konflikteinstellung per Konvention	186
12.6	Fallweise Konflikteinstellung	187
12.7	Zeitstempel (Timestamp)	188
12.8	Konflikte auflösen	190
12.9	Pessimistisches Sperren bei Entity Framework Core	194
13	Protokollierung (Logging)	199
13.1	Verwendung der Erweiterungsmethode Log()	199
13.2	Implementierung der Log()-Erweiterungsmethode	201
13.3	Protokollierungskategorien	204
14	Asynchrone Programmierung	205
14.1	Asynchrone Erweiterungsmethoden	205
14.2	ToListAsync()	205
14.3	SaveChangesAsync()	207
14.4	ForeachAsync()	208
15	Dynamische LINQ-Abfragen	211
15.1	Schrittweises Zusammensetzen von LINQ-Abfragen	211
15.2	Expression Trees	213
15.3	Dynamic LINQ	216
16	Daten lesen und ändern mit SQL, Stored Procedures und Table Valued Functions	219
16.1	Abfragen mit FromSql()	219
16.2	Zusammensetzbarkeit von LINQ und SQL	221
16.3	Globale Abfragefilter bei SQL-Abfragen (ab Version 2.0)	223
16.4	Stored Procedures und Table Valued Functions	224
16.5	Globale Abfragefilter bei Stored Procedures und Table Valued Functions ..	225
16.6	Nicht-Entitätsklassen als Ergebnismenge	226
16.7	SQL-DML-Befehle ohne Resultset	228
17	Weitere Tipps und Tricks zum Mapping	229
17.1	Shadow Properties	229
17.1.1	Automatische Shadow Properties	229
17.1.2	Festlegung eines Shadow Property	230

17.1.3	Ausgabe aller Shadow Properties einer Entitätsklasse	230
17.1.4	Lesen und Ändern eines Shadow Property	231
17.1.5	LINQ-Abfragen mit Shadow Properties	232
17.1.6	Praxisbeispiel: Automatisches Setzen des Shadow Property bei jedem Speichern	232
17.2	Tabellenaufteilung (Table Splitting)	233
17.3	Berechnete Spalten (Computed Columns)	236
17.3.1	Automatisches SELECT	237
17.3.2	Praxistipp: Spalten mit einer Berechnungsformel anlegen	237
17.3.3	Spalten mit einer Berechnungsformel nutzen	239
17.3.4	Spalten mit einer Berechnungsformel beim Reverse Engineering	241
17.4	Standardwerte (Default Values)	242
17.4.1	Standardwerte beim Forward Engineering festlegen	242
17.4.2	Standardwerte verwenden	243
17.4.3	Praxistipp: Standardwerte schon beim Anlegen des Objekts vergeben	245
17.4.4	Standardwerte beim Reverse Engineering	246
17.5	Sequenzobjekte (Sequences)	247
17.5.1	Erstellen von Sequenzen beim Forward Engineering	247
17.5.2	Sequenzen im Einsatz	248
17.6	Alternative Schlüssel	251
17.6.1	Alternative Schlüssel definieren	252
17.6.2	Alternative Schlüssel im Einsatz	254
17.7	Kaskadierendes Löschen (Cascading Delete)	255
17.8	Abbildung von Datenbanksichten (Views)	258
17.8.1	Datenbanksicht anlegen	259
17.8.2	Entitätsklasse für die Datenbanksicht anlegen	259
17.8.3	Einbinden der Entitätsklasse in die Kontextklasse	259
17.8.4	Verwendung der Datenbanksicht	260
17.8.5	Herausforderung: Migrationen	260
18	Weitere Tipps und Tricks zu LINQ	263
18.1	Globale Abfragefilter (ab Version 2.0)	263
18.1.1	Filter definieren	263
18.1.2	Filter nutzen	264
18.1.3	Praxistipp: Filter ignorieren	265
18.2	Zukünftige Abfragen (Future Queries)	265
19	Leistungsoptimierung (Performance Tuning)	267
19.1	Vorgehensmodell zur Leistungsoptimierung bei Entity Framework Core .	267
19.2	Best Practices für Ihre eigenen Leistungstests	268
19.3	Leistungsvergleich verschiedener Datenzugriffstechniken in .NET	268
19.4	Objektzuweisung optimieren	270

19.5	Massenoperationen	272
19.5.1	Einzellöschen	272
19.5.2	Optimierung durch Batching	273
19.5.3	Löschen ohne Laden mit Pseudo-Objekten	274
19.5.4	Einsatz von klassischem SQL anstelle des Entity Framework Core-APIs	275
19.5.5	Lambda-Ausdrücke für Massnlöschen mit EFPlus	277
19.5.6	Massenaktualisierung mit EFPlus	278
19.6	Leistungsoptimierung durch No-Tracking	279
19.6.1	No-Tracking aktivieren	280
19.6.2	No-Tracking fast immer möglich	282
19.6.3	No-Tracking im änderbaren Datagrid	285
19.6.4	QueryTrackingBehavior und AsTracking()	286
19.6.5	Best Practices	288
19.7	Auswahl der besten Ladestrategie	288
19.8	Zwischenspeicherung (Caching)	289
19.8.1	MemoryCache	289
19.8.2	Abstraktion CacheManager	292
19.9	Second-Level-Caching mit EFPlus	296
19.9.1	Einrichten des Second-Level-Cache	297
19.9.2	Verwenden des Second-Level-Cache	297
20	Softwarearchitektur mit Entity Framework Core	301
20.1	Monolithisches Modell	301
20.2	Entity Framework Core als Datenzugriffsschicht	302
20.3	Reine Geschäftslogik	304
20.4	Geschäftsobjekt- und ViewModel-Klassen	305
20.5	Verteilte Systeme	306
20.6	Fazit	309
21	Zusatzwerkzeuge	311
21.1	Entity Framework Core Power Tools	311
21.1.1	Funktionsüberblick	311
21.1.2	Reverse Engineering mit Entity Framework Core Power Tools ...	312
21.1.3	Diagramme mit Entity Framework Core Power Tools	316
21.2	LINQPad	317
21.2.1	Aufbau von LINQPad	318
21.2.2	Datenquellen einbinden	319
21.2.3	LINQ-Befehle ausführen	322
21.2.4	Speichern	325
21.2.5	Weitere LINQPad-Treiber	325
21.2.6	Interaktive Programmcodeeingabe	326
21.2.7	Fazit zu LINQPad	327

21.3	Entity Developer	327
21.3.1	Reverse Engineering mit Entity Developer	328
21.3.2	Forward Engineering mit Entity Developer	339
21.4	Entity Framework Profiler	344
21.4.1	Einbinden des Entity Framework Profilers	346
21.4.2	Befehle überwachen mit Entity Framework Profiler	346
21.4.3	Warnungen vor potenziellen Problemen	349
21.4.4	Analysefunktionen	349
21.4.5	Fazit zu Entity Framework Profiler	350
22	Zusatzkomponenten	351
22.1	Entity Framework Plus (EFPlus)	351
22.2	Second-Level-Caching mit EFSecondLevelCache.Core	352
22.3	Objekt-Objekt-Mapping und AutoMapper	352
22.3.1	Objekt-Objekt-Mapping per Reflection	354
22.3.2	AutoMapper	357
22.3.3	Beispiel	358
22.3.4	Abbildungen konfigurieren	359
22.3.5	Abbildung ausführen mit Map()	360
22.3.6	Abbildungskonventionen	361
22.3.7	Profilklassen	363
22.3.8	Verbundene Objekte	363
22.3.9	Manuelle Abbildungen	364
22.3.10	Typkonvertierungen	366
22.3.11	Objektmengen	368
22.3.12	Vererbung	369
22.3.13	Generische Klassen	371
22.3.14	Zusatzaktionen vor und nach dem Mapping	374
22.3.15	Geschwindigkeit	375
22.3.16	Fazit zu AutoMapper	376
23	Praxislösungen	379
23.1	Entity Framework Core in einer ASP.NET Core-Anwendung	379
23.1.1	Das Fallbeispiel "MiracleList"	379
23.1.2	Architektur	383
23.1.3	Entitätsklassen	385
23.1.4	Entity Framework Core-Kontextklasse	387
23.1.5	Lebensdauer der Kontextklasse in ASP.NET Core-Anwendungen	389
23.1.6	Geschäftslogik	390
23.1.7	WebAPI	398
23.1.8	Verwendung von Entity Framework Core per Dependency Injection	408
23.1.9	Praxistipp: Kontextinstanzpooling (DbContext Pooling)	411
23.2	Entity Framework Core in einer Universal Windows Platform App	412

23.2.1	Das Fallbeispiel "MiracleList Light"	412
23.2.2	Architektur	413
23.2.3	Entitätsklassen	415
23.2.4	Entity Framework Core-Kontextklasse	416
23.2.5	Startcode	416
23.2.6	Erzeugte Datenbank	417
23.2.7	Datenzugriffscodes	419
23.2.8	Benutzeroberfläche	423
23.3	Entity Framework Core in einer Xamarin-Cross-Platform-App	424
23.3.1	Das Fallbeispiel "MiracleList Light"	424
23.3.2	Architektur	426
23.3.3	Entitätsklassen	428
23.3.4	Entity Framework Core-Kontextklasse	428
23.3.5	Startcode	430
23.3.6	Erzeugte Datenbank	430
23.3.7	Datenzugriffscodes	430
23.3.8	Benutzeroberfläche	434
23.4	N:M-Beziehungen zu sich selbst	435
24	Quellen im Internet	441
Index	443

Vorwort

Liebe Leserinnen und Leser,

ich nutze Entity Framework in echten Softwareentwicklungsprojekten seit der allerersten Version, also seit der Version 1.0 von ADO.NET Entity Framework im Jahr 2008. Zuvor hatte ich einen selbstentwickelten Objekt-Relationalen Mapper in meinen Projekten verwendet. Entity Framework Core ist das Nachfolgeprodukt, das es seit 2016 gibt. Ich setzte seitdem auch (aber nicht ausschließlich) Entity Framework Core in der Praxis ein. Viele Projekte laufen noch mit dem klassischen Entity Framework.

Microsoft entwickelt Entity Framework Core inkrementell, d.h. die Versionen 1.x und 2.x stellen zunächst eine in vielen Punkten noch unvollständige Grundversion dar, die in den Folgeversionen dann komplettiert wird.

Dieses **inkrementelle Konzept** habe ich auch mit diesem Buch umgesetzt. Das Buch ist seit September 2016 in mehreren Versionen erschienen. Die vor Ihnen liegende Version dieses Buchs beschreibt nun alle Kernaspekte und viele Tipps und Tricks sowie Praxisszenarien zu Entity Framework Core. Ich plane, in Zukunft weitere Versionen dieses Buchs zu veröffentlichen, die die kommenden Versionen von Entity Framework Core beschreiben und auch weitere Tipps & Tricks sowie Praxisszenarien ergänzen.

Da Fachbücher leider heutzutage nicht nennenswert dazu beitragen können, den Lebensunterhalt meiner Familie zu bestreiten, ist dieses Projekt ein Hobby. Dementsprechend kann ich nicht garantieren, wann es Updates zu diesem Buch geben wird. Ich werde dann an diesem Buch arbeiten, wenn ich neben meinem Beruf als Softwarearchitekt, Berater und Dozent und meinen sportlichen Betätigungen noch etwas Zeit für das Fachbuchautoren-hobby übrig habe.

Zudem möchte ich darauf hinweisen, dass ich natürlich keinen kostenfreien technischen Support zu den Inhalten dieses Buchs geben kann. Ich freue mich aber immer über konstruktives Feedback und Verbesserungsvorschläge. Bitte verwenden Sie dazu das Kontaktformular auf www.dotnet-doktor.de.

Wenn Sie **technische Hilfe** zu Entity Framework und Entity Framework Core oder anderen Themen rund um .NET, Visual Studio, Windows oder andere Microsoft-Produkte benötigen, stehe ich Ihnen im Rahmen meiner beruflichen Tätigkeit für die Firmen www.IT-Visions.de (Beratung, Schulung, Support) und 5Minds IT-Solutions GmbH & Co KG (Softwareentwicklung, siehe www.5minds.de) gerne zur Verfügung. Bitte wenden Sie sich für ein Angebot an das jeweilige Kundenteam.

Die Beispiele zu diesem Buch können Sie herunterladen auf der von mir ehrenamtlich betriebenen **Leser-Website** unter www.IT-Visions.de/Leser. Dort müssen Sie sich registrieren. Bei der Registrierung wird ein Lösungswort abgefragt. Bitte geben Sie dort **Ascension** ein.

Herzliche Grüße aus Essen, dem Herzen der Metropole Ruhrgebiet

Holger Schwichtenberg

Über den Autor



- Studienabschluss Diplom-Wirtschaftsinformatik an der Universität Essen
- Promotion an der Universität Essen im Gebiet komponentenbasierter Softwareentwicklung
- Seit 1996 selbstständig als unabhängiger Berater, Dozent, Softwarearchitekt und Fachjournalist
- Leiter des Berater- und Dozententeams bei www.IT-Visions.de
- Leitung der Softwareentwicklung im Bereich Microsoft/.NET bei der 5minds IT-Solutions GmbH & Co. KG (www.5minds.de)
- Über 65 Fachbücher beim Carl Hanser Verlag, bei O'Reilly, Microsoft Press und Addison-Wesley sowie mehr als 950 Beiträge in Fachzeitschriften
- Gutachter in den Wettbewerbsverfahren der EU gegen Microsoft (2006-2009)
- Ständiger Mitarbeiter der Zeitschriften iX (seit 1999), dotnetpro (seit 2000) und Windows Developer (seit 2010) sowie beim Online-Portal heise.de (seit 2008)
- Regelmäßiger Sprecher auf nationalen und internationalen Fachkonferenzen (z.B. Microsoft TechEd, Microsoft Summit, Microsoft IT Forum, BASTA, BASTA-on-Tour, .NET Architecture Camp, Advanced Developers Conference, Developer Week, OOP, DOTNET



Cologne, MD DevDays, Community in Motion, DOTNET-Konferenz, VS One, NRW.Conf, Net.Object Days, Windows Forum, Container Conf)

- Zertifikate und Auszeichnungen von Microsoft:
- Microsoft Most Valuable Professional (MVP)
- Microsoft Certified Solution Developer (MCSD)
- Thematische Schwerpunkte:
 - Softwarearchitektur, mehrschichtige Softwareentwicklung, Softwarekomponenten, SOA
 - Microsoft .NET Framework, Visual Studio, C#, Visual Basic
 - .NET-Architektur/Auswahl von .NET-Technologien
 - Einführung von .NET Framework und Visual Studio/Migration auf .NET
 - Webanwendungsentwicklung und Cross-Plattform-Anwendungen mit HTML, ASP.NET, JavaScript/TypeScript und Webframeworks wie Angular
 - Enterprise .NET, verteilte Systeme/Webservices mit .NET, insbesondere Windows Communication Foundation und WebAPI
 - Relationale Datenbanken, XML, Datenzugriffsstrategien
 - Objektrelationales Mapping (ORM), insbesondere ADO.NET Entity Framework und EF Core
 - Windows PowerShell, PowerShell Core und Windows Management Instrumentation (WMI)
- Ehrenamtliche Community-Tätigkeiten:
 - Vortragender für die International .NET Association (INETA)
 - Betrieb diverser Community-Websites: www.dotnetframework.de, www.entwicklerlexikon.de, www.windows-scripting.de, www.aspnetdev.de u. a.
- Firmenwebsites: <http://www.IT-Visions.de> und <http://www.5minds.de>
- Weblog: <http://www.dotnet-doktor.de>
- Kontakt: E-Mail buero@IT-Visions.de sowie Telefon **0201-64 95 90-0**

1

Einleitung

■ 1.1 Programmiersprache in diesem Buch

Als Programmiersprache kommt in diesem Buch C# zum Einsatz, weil dies die bei weitem am häufigsten verwendete Programmiersprache in .NET ist. Der Autor dieses Buchs programmiert in einigen Kundenprojekten .NET-Anwendungen zwar auch in Visual Basic .NET, leider bietet dieses Buch jedoch nicht den Raum, alle Listings in beiden Sprachen wiederzugeben.

Eine Sprachkonvertierung zwischen C# und Visual Basic .NET ist im WWW kostenfrei verfügbar auf der Website <http://converter.telerik.com>.

■ 1.2 Fallbeispiele in diesem Buch

Die meisten Beispielprogrammcodes in diesem Buch drehen sich um das Fallbeispiel der fiktiven Fluggesellschaft „World Wide Wings“, abgekürzt WWWings (siehe auch www.world-wide-wings.de).



Bild 1.1 Logo der fiktiven Fluggesellschaft „World Wide Wings“



HINWEIS: In einzelnen Unterkapiteln werden andere Fallbeispiele verwendet (z. B. die Aufgabenverwaltung „MiracleList“). Diese Fallbeispiele werden dann in den jeweiligen Kapiteln erläutert.

1.2.1 Entitäten

Im Anwendungsfall „World Wide Wings“ geht es um folgende Entitäten:

- Flüge zwischen zwei Orten, bei denen die Orte bewusst nicht als eigene Entität modelliert wurden, sondern Zeichenketten sind (dies vereinfacht das Verständnis vieler Beispiele)
- Passagiere, die auf Flügen fliegen
- Mitarbeiter der Fluggesellschaft, die wiederum Vorgesetzte haben, die auch Mitarbeiter sind
- Piloten als eine Spezialisierung von Mitarbeitern
- Personen als Sammlung der gemeinsamen Eigenschaften für alle Menschen in diesem Beispiel. Personen gibt es aber nicht eigenständig, sondern nur in den Ausprägungen/Spezialisierungen Passagier, Mitarbeiter und Pilot. Im objektorientierten Sinne ist Person also eine abstrakte Basisklasse, die keine Instanzen besitzen kann, sondern nur der Vererbung dient.

Es gibt zwei Datenmodelle:

- Das etwas einfachere Modell #1 ist das Ergebnis klassischen relationalen Datenbankdesigns mit Normalisierung. Das Objektmodell daraus entsteht per Reverse Engineering.
- Modell #2 ist das Ergebnis des Forward Engineering mit Entity Framework Core aus einem Objektmodell. Zusätzlich gibt es hier weitere Entitäten (Persondetail, Flugzeugtyp und Flugzeugtypdetail), um weitere Modellierungsaspekte aufzeigen zu können.

In Modell #1 gibt es eine jeweils eigene Tabelle für Personen (auch wenn es keine eigenständigen Personen gibt), Mitarbeiter, Piloten und Passagiere. Diese Aufteilung entspricht den Klassen im Objektmodell.

In Modell #2 gibt es lediglich die Tabellen Passagiere und Mitarbeiter für diese vier Entitäten. Entity Framework Core ist derzeit etwas eingeschränkt und unterstützt das Konzept Table per Type (also eine eigenständige Tabelle für jede Klasse) nicht. Daher umfasst die Tabelle Passagiere auch alle Eigenschaften von Person. Die Tabelle Mitarbeiter umfasst neben den Personeneigenschaften die Eigenschaften der Entitäten Mitarbeiter und Pilot. In der Tabelle wird per Diskriminatorspalte unterschieden zwischen Datensätzen, die ein Mitarbeiter sind, und solchen, die ein Pilot sind. Entity Framework Core mischt hier die Konzepte Table per Concrete Type (TPC) und Table per Hierarchy (TPH). Einen dezidierten Einfluss auf diese Abbildung hat man in Entity Framework Core 1.x/2.0 noch nicht. Das klassische Entity Framework bietet hier mehr Optionen.

Abhängigkeitsarten sind hier:

- Ein Flug muss einen Piloten besitzen. Der Copilot ist optional.
- Ein Flug kann optional einen Flugzeugtyp zugeordnet haben.
- Jede Person und damit auch jeder Pilot und Passagier muss ein Persondetail-Objekt besitzen.

In diesem Buch kommen beide Datenmodelle vor, teilweise auch in modifizierter Form, um bestimmte Szenarien (z. B. Datenbankschemamigrationen) aufzuzeigen.

Bitte beachten Sie, dass die Objektmodelle, die in diesem Buch zu den Datenmodellen erstellt werden, nicht das Idealbild eines Objektmodells darstellen können, denn Entity Framework Core unterstützt einige Mapping-Möglichkeiten wie z. B. das N:M-Mapping noch nicht. Das Objektmodell zum einfachen Datenmodell ist das automatisch von Entity Framework Core aus der Datenbank generierte Objektmodell (Reverse Engineering); es ist bewusst nicht verändert worden.

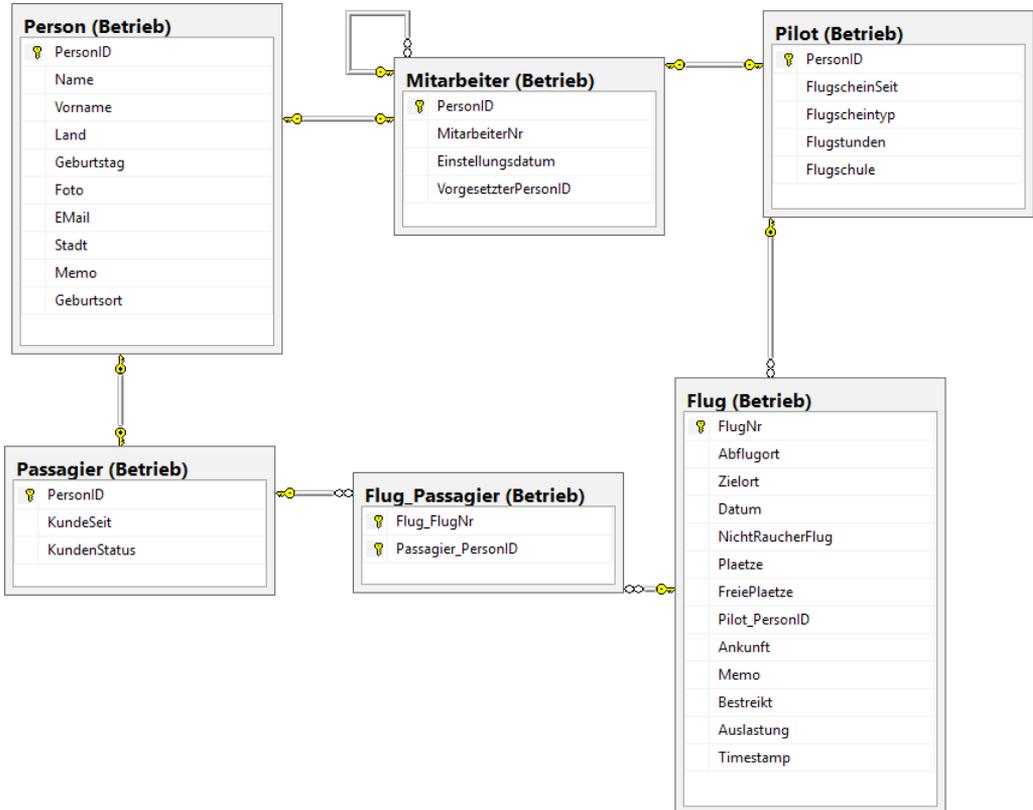


Bild 1.2 World Wide Wings-Datenmodell in der einfacheren Version

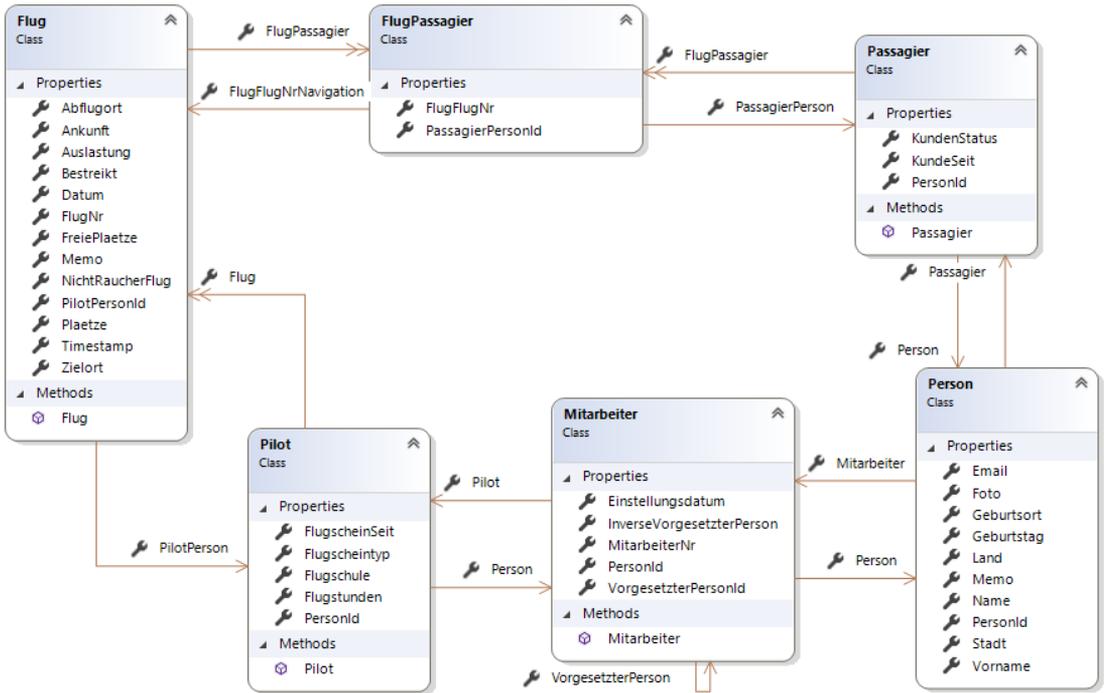


Bild 1.3 Objektmodell zum World Wide Wings-Datenmodell in der einfacheren Version

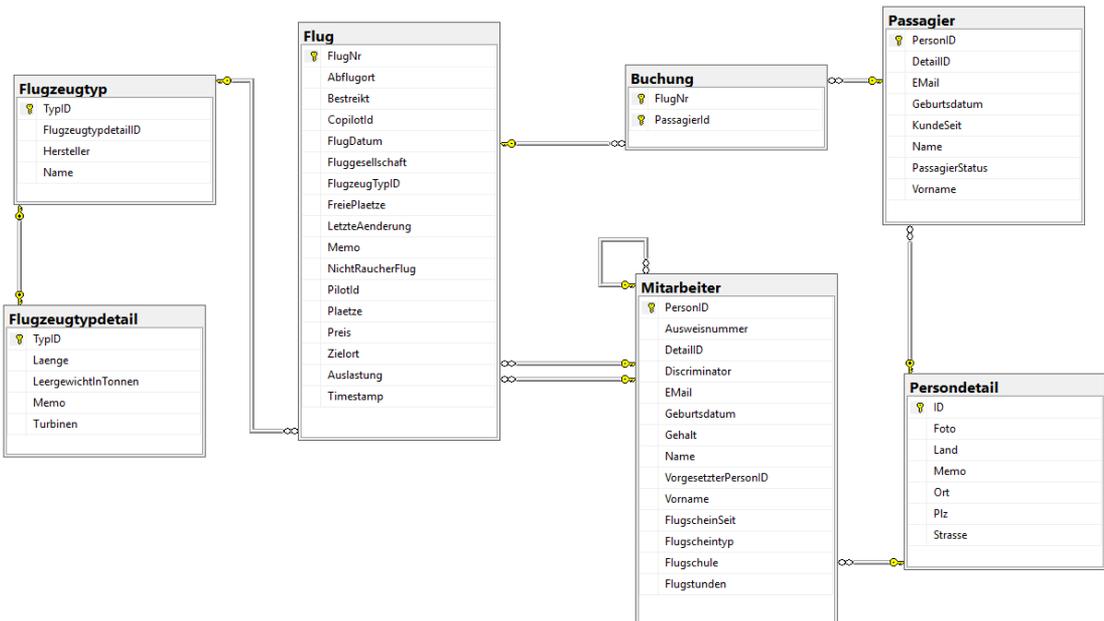


Bild 1.4 World Wide Wings-Datenmodell in der komplexeren Version

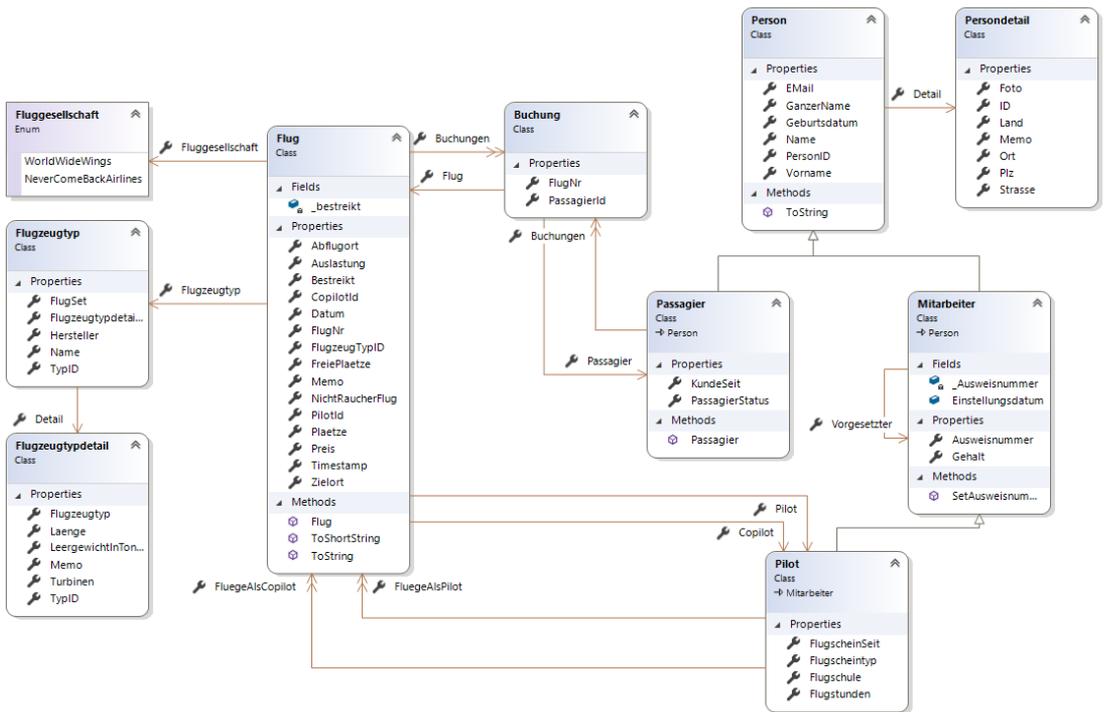


Bild 1.5 Objektmodell zum World Wide Wings-Datenmodell in der komplexeren Version

1.3 Anwendungsarten in diesem Buch

In diesem Buch erfolgen Bildschirmausgaben meist an der textbasierten Konsole in Konsolenanwendungen, denn dies ermöglicht die Fokussierung auf den Datenbankzugriff. Beim Einsatz von grafischen Benutzeroberflächen wie WPF, Windows Forms, ASP.NET Webforms oder ASP.NET MVC ist die Darstellung durch Datenbindung entkoppelt, das heißt man würde immer ein zweites Listing brauchen, um zu verstehen, dass die Datenzugriffe überhaupt liefern. Eingaben des Benutzers werden in den Konsolenbeispielen durch Variablen zu Beginn des Programmcodes simuliert.

Der Autor dieses Buchs führt seit vielen Jahren Schulungen und Beratungseinsätze im Bereich Datenzugriff durch und hat dabei die Erfahrung gemacht, dass Konsolenausgaben das didaktisch beste Instrument sind, da die Listings sonst sehr umfangreich und damit schlechter zu verstehen sind.

Natürlich ist die Konsolenausgabe in 99% der Fälle der Softwareentwicklung nicht die gängige Praxis. Grafische Benutzeroberflächen sind Inhalt anderer Bücher, und die Datenbindung hat in der Regel keinen Einfluss auf die Form des Datenzugriffs. Dort, wo der Datenzugriff doch relevant ist, wird dieses Buch auch Datenbindungsbeispiele zeigen.

■ 1.4 Hilfsroutinen zur Konsolenausgabe

Für die Bildschirmausgabe an der Konsole wird an mehreren Stellen nicht nur `Console.WriteLine()` verwendet, sondern auch Hilfsroutinen kommen zur Anwendung, die farbige Bildschirmausgaben erzeugen. Diese Hilfsroutinen in der Klasse `CUI` aus der `ITV_DemoUtil.dll` sind hier zum besseren Verständnis abgedruckt:

Listing 1.1 Klasse `CUI` mit Hilfsroutinen für die Bildschirmausgabe an der Konsole

```
using System;
using System.Runtime.InteropServices;
using System.Web;
using ITVisions.UI;
using System.Diagnostics;

namespace ITVisions
{
    /// <summary>
    /// Hilfsroutinen für Konsolen-UIs
    /// (C) Dr. Holger Schwichtenberg 2002-2017
    /// </summary>
    public static class CUI
    {
        public static bool IsDebug = false;
        public static bool IsVerbose = false;

        #region Ausgaben unter bestimmten Bedingungen
        /// <summary>
        /// Ausgabe an Console, Trace und Datei - nur wenn Anwendung im DEBUG-Modus
        /// </summary>
        public static void PrintDebug(object s)
        {
            PrintDebug(s, System.Console.ForegroundColor);
        }
        /// <summary>
        /// Ausgabe an Console, Trace und Datei - nur wenn Anwendung im VERBOSE-Modus
        /// </summary>
        public static void PrintVerbose(object s)
        {
            PrintVerbose(s, System.Console.ForegroundColor);
        }
        #endregion

        #region Ausgaben mit vordefinierten Farben
        public static void MainHeadline(string s)
        {
            Print(s, ConsoleColor.Black, ConsoleColor.Yellow);
        }

        public static void Headline(string s)
        {
            Print(s, ConsoleColor.Yellow);
        }

        public static void HeaderFooter(string s)
        {

```

```
Console.ForegroundColor = ConsoleColor.Green;
Console.WriteLine(s);
Console.ForegroundColor = ConsoleColor.Gray;
}

public static void PrintSuccess(object s)
{
    Print(s, ConsoleColor.Green);
}

public static void PrintDebugSuccess(object s)
{
    PrintDebug(s, ConsoleColor.Green);
}

public static void PrintVerboseSuccess(object s)
{
    PrintVerbose(s, ConsoleColor.Green);
}

public static void PrintWarning(object s)
{
    Print(s, ConsoleColor.Cyan);
}

public static void PrintDebugWarning(object s)
{
    PrintDebug(s, ConsoleColor.Cyan);
}

public static void PrintVerboseWarning(object s)
{
    PrintVerbose(s, ConsoleColor.Cyan);
}

public static void PrintError(object s)
{
    Print(s, ConsoleColor.White, ConsoleColor.Red);
}

public static void PrintDebugError(object s)
{
    PrintDebug(s, ConsoleColor.White, ConsoleColor.Red);
}

public static void PrintVerboseError(object s)
{
    Print(s, ConsoleColor.White, ConsoleColor.Red);
}

public static void Print(object s)
{
    PrintInternal(s, null);
}
```

```

}
#endregion

#region Ausgaben mit wählbarer Farbe

/// <summary>
/// Ausgabe an Console, Trace und Datei - nur wenn Anwendung im DEBUG Modus
/// </summary>
public static void Print(object s, ConsoleColor farbe, ConsoleColor?
hintergrundfarbe = null)
{
    PrintInternal(s, farbe, hintergrundfarbe);
}

public static void PrintDebug(object s, ConsoleColor farbe, ConsoleColor?
hintergrundfarbe = null)
{
    if (IsDebug || IsVerbose) PrintDebugOrVerbose(s, farbe, hintergrundfarbe);
}

public static void PrintVerbose(object s, ConsoleColor farbe)
{
    if (!IsVerbose) return;
    PrintDebugOrVerbose(s, farbe);
}
#endregion

#region Spezielle Ausgabe mit Zusatzdaten

/// <summary>
/// Ausgabe mit Thread-ID
/// </summary>
public static void PrintWithThreadID(string s, ConsoleColor c = ConsoleColor.White)
{
    var ausgabe = String.Format("Thread #{0:00} {1:}: {2}", System.Threading.Thread.
CurrentThread.ManagedThreadId, DateTime.Now.ToLongTimeString(), s);
    CUI.Print(ausgabe, c);
}

/// <summary>
/// Ausgabe mit Uhrzeit
/// </summary>
public static void PrintWithTime(object s, ConsoleColor c = ConsoleColor.White)
{
    CUI.Print(DateTime.Now.Second + "." + DateTime.Now.Millisecond + ":" + s);
}

private static long count;
/// <summary>
/// Ausgabe mit fortlaufendem Zähler
/// </summary>
private static void PrintWithCounter(object s, ConsoleColor farbe, ConsoleColor?
hintergrundfarbe = null)
{
    count += 1;
    s = String.Format("{0:0000}: {1}", count, s);
    CUI.Print(s, farbe, hintergrundfarbe);
}

```

```
#endregion

#region interne Hilfsroutinen
private static void PrintDebugOrVerbose(object s, ConsoleColor farbe, ConsoleColor?
hintergrundfarbe = null)
{
    count += 1;
    s = String.Format("{0:0000}: {1}", count, s);
    Print(s, farbe, hintergrundfarbe);
    Debug.WriteLine(s);
    Trace.WriteLine(s);
    Trace.Flush();
}

/// <summary>
/// Ausgabe an Console, Trace und Datei
/// </summary>
/// <param name="s"></param>
/// [DebuggerStepThrough]
private static void PrintInternal(object s, ConsoleColor? farbe = null,
ConsoleColor? hintergrundfarbe = null)
{
    if (s == null) return;

    if (HttpContext.Current != null)
    {
        try
        {
            //Achtung: Keine Textausgaben in Ausgabestrom von ASP.NET Webservices und WCF-
            //Diensten und WebAPI schreiben!
            if (farbe != null)
            {
                HttpContext.Current.Response.Write("<span style='color:" + farbe.Value.
                DrawingColor().Name + "'>");
            }
            if (!HttpContext.Current.Request.Url.ToString().ToLower().Contains(".asmx") &&
            !HttpContext.Current.Request.Url.ToString().ToLower().Contains(".svc") &&
            !HttpContext.Current.Request.Url.ToString().ToLower().Contains("/api/")) HttpContext.
            Current.Response.Write(s.ToString() + "<br>");

            if (farbe != null)
            {
                HttpContext.Current.Response.Write("</span>");
            }
        }
        catch (Exception)
        {
        }
    }
    else
    {
        object x = 1;
        lock (x)
        {
            ConsoleColor alteFarbe = Console.ForegroundColor;
            ConsoleColor alteHFarbe = Console.BackgroundColor;

```

```

        if (farbe != null) Console.ForegroundColor = farbe.Value;
        if (hintergrundfarbe != null) Console.BackgroundColor = hintergrundfarbe.Value;

        //if (farbe.ToString().Contains("Dark")) Console.BackgroundColor = ConsoleColor.
White;
        //else Console.BackgroundColor = ConsoleColor.Black;

        Console.WriteLine(s);
        Console.ForegroundColor = alteFarbe;
        Console.BackgroundColor = alteHFarbe;
    }
}
}
#endregion

#region Position des Konsolenfensters setzen
[DllImport("kernel32.dll", ExactSpelling = true)]
private static extern IntPtr GetConsoleWindow();
private static IntPtr MyConsole = GetConsoleWindow();

[DllImport("user32.dll", EntryPoint = "SetWindowPos")]
public static extern IntPtr SetWindowPos(IntPtr hWnd, int hWndInsertAfter, int x,
int Y, int cx, int cy, int wFlags);

// Setze Position des Konsolenfensters ohne Größe
public static void SetConsolePos(int xpos, int ypos)
{
    const int SWP_NOSIZE = 0x0001;
    SetWindowPos(MyConsole, 0, xpos, ypos, 0, 0, SWP_NOSIZE);
}

// Setze Position des Konsolenfensters mit Größe
public static void SetConsolePos(int xpos, int ypos, int w, int h)
{
    SetWindowPos(MyConsole, 0, xpos, ypos, w, h, 0);
}
#endregion
}
}
}

```

■ 1.5 Programmcodebeispiel zum Download

Die Beispiele zu diesem Buch können Sie als Visual Studio-Projekte herunterladen auf der Leser-Website unter www.IT-Visions.de/Leser. Dort müssen Sie sich einmalig registrieren. Bei der Registrierung wird ein **Losungswort** abgefragt, das Sie als Käufer dieses Buchs ausweist. Bitte geben Sie dort **Ascension** ein. Durch die Registrierung erhalten Sie ein persönliches Kennwort per E-Mail zugesendet, das Sie dann für die Anmeldung nutzen können. Bitte beachten Sie, dass nicht jede einzelne Zeile Programmcode, die Sie in diesem Buch finden, in den herunterladbaren Projekten enthalten sein kann. Die Projekte bilden funkti-