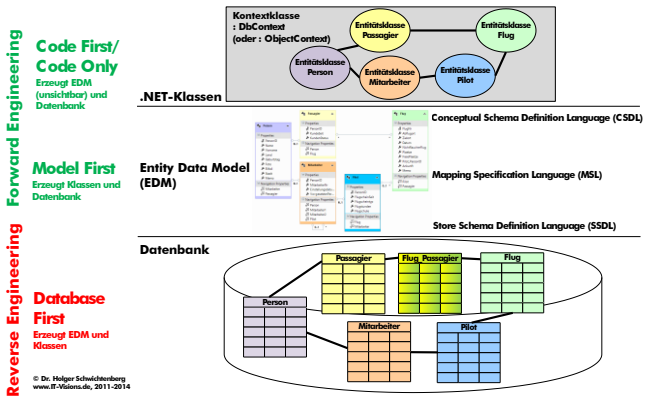
Kurzreferenz („Cheat Sheet“) ADO.NET Entity Framework mit DbContext

Autor: Dr. Holger Schwichtenberg (www.IT-Visions.de)

Artefakte, Vorgehensweisen und Kontextarten



Kontextbasisklasse	ObjectContext	DbContext
EF-Versionen	1.0 bis 6.x (nicht in Core 1.0)	Ab 4.1
Code First	Nicht möglich	möglich & empfohlen
Model First	möglich	möglich & empfohlen
Database First	möglich	möglich & empfohlen

ObjectContext wird in einem anderen Cheat Sheet behandelt.

Werkzeuge zur Modellerstellung

- Microsoft Visual Studio: Vorlage „ADO.NET Entity Data Model“
- Microsoft Kommandozeilenwerkzeug edmgen.exe
- DevArt Entity Developer

Die Modellerstellung für Entity Framework Code First/Code Only wird auf einen eigenen Cheat Sheet behandelt.

Benötigte Namensräume für using-Befehle

Diese Namensräume werden typischerweise gebraucht:
System.Collections.Generic, System.Linq, System.Data, System.Data.Entity, System.Data.Entity.SqlServer, System.Data.Entity.Infrastructure

Ladeoperationen

Laden eines Datensatzes anhand des Primärschlüssels mit **Find()**

```
using (WWWings6Entities ctx = new WWWings6Entities())
{  int flugNr = 123;
    var f = ctx.Flug.Find(flugNr);
   Console.WriteLine("Flug: " + f.FlugNr + " von " + f.Abflugort + " nach " +
   f.Zielort + " hat " + f.FreiePlaetze + " freie Plätze");
}
```

Laden aller Datensätze einer Tabelle

```
using (WWWings6Entities ctx = new WWWings6Entities())
{  var liste = ctx.Flug.ToList();
   foreach (var f in liste)  { Console.WriteLine("Flug: " + f.FlugNr + …)  }
}
```

Wichtige LINQ-Befehle

Bedingungen: Laden aller Objekte

```
var liste = (from f in ctx.Flug
   where (f.Abflugort == "Rom" && f.FreiePlaetze > 5) ||
   f.Passagier.Any(p => p.Person.Name == "Meier")
   select f).ToList();
```

Bedingungen: Laden des ersten Objekts

```
var flug = (from f in ctx.Flug
   where (f.Abflugort == "Rom" && f.FreiePlaetze > 5) ||
      f.Passagier.All(p => p.Person.Geburtstag >
   DbFunctions.AddYears(DateTime.Now, -18))
   select f).FirstOrDefault();
```

Sortieren

```
var liste = (from f in ctx.Flug
   orderby f.Datum descending, f.Abflugort, f.FreiePlaetze
   select f).ToList();
```

Paging: Datensätze 1201 bis 1210 (Sortieren ist dafür erforderlich!)

```
var liste = (from f in ctx.Flug orderby f.FlugNr
select f).Skip(1200).Take(10).ToList();
```

Gruppieren flach

```
var gruppe = (from f in ctx.Flug
   group f by f.Abflugort into g
   select new { Abflugort = g.Key, Anzahl = g.Count(), FreiePlaetze = g.Sum(f
   => f.FreiePlaetze) }).ToList();
foreach (var g in gruppe)
{
   Console.WriteLine(g.Anzahl + " Flüge von " + g.Abflugort + " mit " +
   g.FreiePlaetze + " freien Plätzen");
}
```

Gruppieren hierarchisch

```
var gruppe = (from f in ctx.Flug
   group f by f.Abflugort into g
   select new { Abflugort = g.Key, Anzahl = g.Count(), Fluege = g }).ToList();
foreach (var g in gruppe)
{
Console.WriteLine(g.Anzahl + " Flüge von " + g.Abflugort);
foreach (var f in g.Fluege)
   {
   Console.WriteLine("- Flug: " + f.FlugNr + " von " + f.Abflugort + " nach " +
   f.Zielort + " hat " + f.FreiePlaetze + " freie Plätze");
   }
}
```

Direktes SQL

SQL-Abfrage oder SP, die Resultset liefert (Entitätstyp oder andere Klasse)

```
string Ort1 = "Paris";
string Ort2 = "Rom";
var liste = ctx.Database.SqlQuery<Flug>(
"Select * from Flug where Abflugort = {0} and Zielort = {1}
order by FreiePlaetze desc", Ort1, Ort2).ToList();
```

SQL-Abfrage, die primitive Daten liefert

```
var abflugorte = ctx.Database.SqlQuery<string>("select distinct Abflugort from
Flug").ToList();
foreach (var ort in abflugorte) { Console.WriteLine(ort);  }
```

SQL-DML-Anweisung, die Daten löscht

```
var anzahl = ctx.Database.ExecuteSqlCommand(
"delete from Flug where flugNr < 100");
Console.WriteLine("Anzahl gelöschter Flüge: " + anzahl);
```

CUD-Operationen

Objekt im RAM anlegen und persistieren

```
var flug = ctx.Flug.Create(); // nicht new Flug() verwenden!
flug.FlugNr = 99;
flug.Abflugort = "Essen/Mülheim";
flug.Zielort = "Redmond";
flug.Datum = new DateTime(2014, 8, 1);
ctx.Flug.Add(flug);
var anzahl = ctx.SaveChanges();
Console.WriteLine("Gespeicherte Änderungen: " + anzahl.ToString());
```

Objekt im RAM ändern und persistieren

```
var flug = ctx.Flug.Find(99);
flug.FreiePlaetze--;
flug.Datum = flug.Datum.AddHours(2);
var anzahl = ctx.SaveChanges();
Console.WriteLine("Gespeicherte Änderungen: " + anzahl.ToString());
```

Objekt im RAM löschen und das Löschen persistieren

```
int flugNr = 99;
var flug = ctx.Flug.Where(x => x.FlugNr == flugNr).SingleOrDefault();
if (flug != null) ctx.Flug.Remove(flug);
var anzahl = ctx.SaveChanges();
Console.WriteLine("Gespeicherte Änderungen: " + anzahl.ToString());
```

Transaktionen mit System.Transactions

SaveChanges() erzeugt immer automatisch eine Transaktion. Es ist möglich, mehrere SaveChanges()-Aufrufe zu einer Transaktion zusammenzufassen.

```
using (WWWings6Entities ctx = new WWWings6Entities())
{
 var tso = System.Transactions.TransactionScopeOption.Required;
 var to = new System.Transactions.TransactionOptions();
 to.IsolationLevel = IsolationLevel.Snapshot;
 using (var t = new System.Transactions.TransactionScope(tso, to))
 {
  var flug1 = ctx.Flug.Find(99);
  flug1.FreiePlaetze--;
  var anzahl1 = ctx.SaveChanges();
  Console.WriteLine("Gespeicherte Änderungen: " + anzahl1.ToString());
  var flug2 = ctx.Flug.Find(100);
  flug2.FreiePlaetze--;
  var anzahl2 = ctx.SaveChanges();
  Console.WriteLine("Gespeicherte Änderungen: " + anzahl2.ToString());
  t.Complete(); // Sonst kommt es zum Rollback!
  Console.WriteLine("Transaktion erfolgreich!");
 }
}
```

Kurzreferenz („Cheat Sheet“) ADO.NET Entity Framework mit DbContext

Autor: Dr. Holger Schwichtenberg (www.IT-Visions.de)

Konflikterkennung und -lösung

```
try
{ int anzahl = ctx.SaveChanges(); }
catch (DbUpdateConcurrencyException ex)
{
var flugAnderer = (Flug)ex.Entries.Single().GetDatabaseValues().ToObject();
Console.WriteLine(DateTime.Now.ToLongTimeString() + ": Ein andere Benutzer hat
den Flug bereits geändert. Freie Plätze dort:" + flugAnderer.FreiePlaetze);
}
```

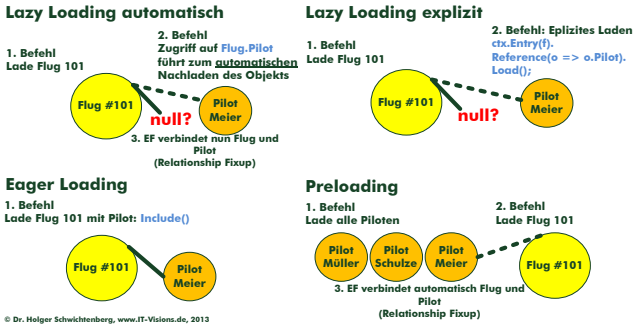
Werte des anderen Benutzers übernehmen

```
ctx.Entry(flugDieser).Reload();
```

Werte des anderen Benutzers überschreiben

```
ctx.Entry(flugDieser).OriginalValues.SetValues(ctx.Entry(flugDieser).GetDatabaseVa
lues();
int anzahl = ctx.SaveChanges();
```

Ladestrategien



Automatisches Lazy Loading: Zugriff auf Pilot und seine Detaildaten ist möglich, führt aber jeweils zu einer eigenen Abfrage.

```
var fluege = (from f in ctx.Flug where f.Abflugort == "Rom" select f);
foreach (var f in fluege.ToList())
{
   Console.WriteLine("Flug: " + f.FlugNr + " von " + f.Abflugort …);
   Console.WriteLine("Pilot: " + f.Pilot.Mitarbeiter.Person.Name);
}
```

Explizites Lazy Loading: Zugriff auf Pilot und seine Detaildaten ist nur möglich mit expliziten Load()-Anweisungen für jede einzelne Ebene.

```
var fluege = (from f in ctx.Flug where f.Abflugort == "Rom" select f);
ctx.Configuration.LazyLoadingEnabled = false;
foreach (var f in fluege.ToList())
{
if (!ctx.Entry(f).Reference(o => o.Pilot).IsLoaded)
ctx.Entry(f).Reference(o => o.Pilot).Load();
if (!ctx.Entry(f.Pilot).Reference(o => o.Mitarbeiter).IsLoaded)
ctx.Entry(f.Pilot).Reference(o => o.Mitarbeiter).Load();
if (!ctx.Entry(f.Pilot.Mitarbeiter).Reference(o => o.Person).IsLoaded)
ctx.Entry(f.Pilot.Mitarbeiter).Reference(o => o.Person).Load();
Console.WriteLine("Pilot: " + f.Pilot.Mitarbeiter.Person.Name);
}
```

Eager Loading: Pilot und seine Detaildaten werden direkt zusammen mit den Flügen geladen

```
var fluege = (from f in ctx.Flug
   .Include(x=>x.Pilot.Mitarbeiter.Person)
   where f.Abflugort == "Rom"
   select f);
foreach (var f in fluege.ToList())
{
   Console.WriteLine("Flug: " + f.FlugNr + " von " + f.Abflugort …);
   Console.WriteLine("Pilot: " + f.Pilot.Mitarbeiter.Person.Name);
}
```

Preloading: Alle Piloten und alle Details der Piloten werden vorab geladen und werden beim Laden der Flüge automatisch mit den Flügen verbunden.

```
// Alle Piloten laden
ctx.Pilot.ToList();
// Alle Mitarbeiterdetails zu allen Piloten laden
ctx.Mitarbeiter.Where(x => x.Pilot != null).ToList();
 // Alle Personendetails zu allen Piloten laden
ctx.Person.Where(x => x.Mitarbeiter.Pilot != null).ToList();
var fluege = (from f in ctx.Flug where f.Abflugort == "Rom" select f);
foreach (var f in fluege.ToList())
{
   Console.WriteLine("Flug: " + f.FlugNr + " von " + f.Abflugort …);
   Console.WriteLine("Pilot: " + f.Pilot.Mitarbeiter.Person.Name);
}
```

Optimierung durch NoTracking

Laden ohne Änderungsverfolgung ist deutlich schneller!

```
var liste = (from f in ctx.Flug.AsNoTracking() orderby f.FlugNr select f).ToList();
```

Nachträgliches **Attach()** erlaubt dann doch Änderungsverfolgung:

```
var gewaehlterFlug = liste.ElementAt(10);
ctx.Flug.Attach(gewaehlterFlug);
gewaehlterFlug.FreiePlaetze--;
int anzahl = ctx.SaveChanges();
```

Protokollierung (ab EF 6.0)

```
ctx.Database.Log = LogToConsole; // Verweis auf Methode mit string-Parameter
…
public static void LogToConsole(string s)
{
if (!s.StartsWith("-"))
   { Console.ForegroundColor = ConsoleColor.Yellow; }
else
   { Console.ForegroundColor = ConsoleColor.Green; }
Console.Write(s);
}
```

Weitergehende Beeinflussung der Protokollierung ist möglich mit einer von **DatabaseLogFormatter** abgeleiteten Klasse oder einer Implementierung der Schnittstelle **IDbCommandInterceptor**.

Validierung (System.Data.Entity.Validation)

Ermitteln der Überprüfungsfehlerliste (jederzeit möglich)

```
var fehlerliste = ctx.GetValidationErrors();
```

Abfangen von Prüfungsfehlern beim Speichern

```
try
   { int anzahl = ctx.SaveChanges();}
catch (DbEntityValidationException ex)
   { var fehlerliste = ex.EntityValidationErrors;      }
```

Abschalten der Prüfung beim Speichern

```
ctx.Configuration.ValidateOnSaveEnabled = false;
```

Ausgabe der Fehlerliste

```
int count = 0;  string ausgabe = "";
foreach (var efObj in fehlerListe)
{  count++;
   ausgabe += count + ". fehlerhaftes EF-Objekt " + efObj.Entry.Entity + "\n";
   foreach (DbValidationError eintrag in efObj.ValidationErrors)
      { ausgabe+= eintrag.PropertyName + ": " + eintrag.ErrorMessage  + "\n"; }
}
```

Links

Beispieldatenbank „World Wide Wings“ Version 6
http://www.world-wide-wings.de
Offizielle Microsoft-Seite im Data Developer Center:
http://msdn.microsoft.com/en-us/data/ef.aspx
Tutorial:
http://www.entityframeworktutorial.net
Projektwebsite:
http://entityframework.codeplex.com
Team Blog:
http://blogs.msdn.com/b/adonet
Entity Framework-Quellcode:
git clone https://git01.codeplex.com/entityframework.git
Entity Framework-Kompilat:
http://www.nuget.org/packages/EntityFramework
http://www.nuget.org/packages/EntityFramework.SqlServerCompact
Entity Framework Power Tools für Visual Studio:
http://bit.ly/1cdobhk
Entity Framework Profiler:
http://www.efprof.com
LINQPad:
http://www.linqpad.net
DevArt Entity Developer und Entity Framework-Provider für diverse DBMS:
http://www.devart.com

Über den Autor

Dr. Holger Schwichtenberg gehört zu den bekanntesten Experten für die Programmierung mit Microsoft-Produkten in Deutschland. Er hat zahlreiche Fachbücher zu .NET, ASP.NET und PowerShell veröffentlicht und spricht regelmäßig auf Fachkonferenzen wie der BASTA. Sie können ihn für Schulungen, Beratungen und Projekte buchen.
E-Mail: hs@IT-Visions.de ▪ Telefon: +49 201 649590-40
Website: www.IT-Visions.de ▪ Weblog: www.dotnet-doktor.de