

Entity Framework

Notes for Professionals

Chapter 2: Code First Conventions

Section 2.1: Removing Conventions

You can remove any of the conventions defined in the `System.Data.Entity.ModelConfiguration` namespace, by overriding `OnModelCreating` method.

The following example removes `PluralizingTableNameConvention`.

```
public class EshopContext : DbContext
{
    public DbSet<Product> Products { get; set; }
    ...
    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        modelBuilder.Conventions.Remove<PluralizingTableNameConvention>();
    }
}
```

By default EF will create DB table with entity class name suffixed by 's'. In this example ignore `PluralizingTableNameConvention` so, instead of `dbo.Products` table `dbo.Product` will be created.

Section 2.2: Primary Key Convention

By default a property is a primary key if a property on a class is named "ID" (not case followed by "ID", if the type of the primary key property is numeric or GUID it will be column. Simple Example:

```
public class Room
{
    // Primary key
    public int RoomId { get; set; }
    ...
}
```

Section 2.3: Type Discovery

By default Code First includes in model

1. Types defined as a DbSet property in context class.
2. Reference types included in every types even if they are defined in its base class.
3. Derived classes even if only the base class is defined as DbSet property.

Here is an example, that we are only adding company as DbSet<Company>.

```
public class Company
{
    public int ID { get; set; }
    public string Name { get; set; }
    public virtual ICollection<Department> Departments { get; set; }
}

public class Department
{
    public int ID { get; set; }
    ...
}
```

Chapter 3: Code First DataAnnotations

Section 3.1: [Column] attribute

```
public class Person
{
    public int PersonID { get; set; }
    [Column("NameOfPerson")]
    public string PersonName { get; set; }
}
```

Tells Entity Framework to use a specific column name instead using the name of the property. You can also specify the database data type and the order of the column in table:

```
[Column("NameOfPerson", TypeName = "varchar", order = 1)]
public string PersonName { get; set; }
```

Section 3.2: [DatabaseGenerated] attribute

Specifies how the database generates values for the property. There are three possible values:

1. None specifies that the values are not generated by the database.
2. Identity specifies that the column is an **identity column**, which is typically used for integer primary keys.
3. Computed specifies that the database generates the value for the column.

If the value is anything other than None, Entity Framework will not commit changes made to the property back to the database.

By default (based on the `StoreGeneratedIdentityKeyConvention`) an integer key property will be treated as an identity column. To override this convention and force it to be treated as a non-identity column you can use the `DatabaseGenerated` attribute with a value of `None`.

```
using System.ComponentModel.DataAnnotations.Schema;
```

```
public class Foo
{
    [Key]
    public int ID { get; set; } // identity (auto-increment) column
}

public class Bar
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.None)]
    public int ID { get; set; } // non-identity column
}
```

The following SQL creates a table with a computed column:

```
CREATE TABLE [Person] (
    Name varchar (100) PRIMARY KEY,
    DateOfBirth Date NOT NULL,
    Age AS DATEDIFF(year, DateOfBirth, GETDATE())
)
```

Entity Framework Notes for Professionals

Chapter 14: Transactions

Section 14.1: Database.BeginTransaction()

Multiple operations can be executed against a single transaction so that changes can be rolled back if any of the operations fail.

```
using (var context = new PlanetContext())
using (var transaction = context.Database.BeginTransaction())
{
    try
    {
        //lets show how this works
        var jupiter = new Planet { Name = "Jupiter" };
        context.Planets.Add(jupiter);
        context.SaveChanges();

        //And then this will throw an exception
        var neptune = new Planet { Name = "Neptune" };
        context.Planets.Add(neptune);
        context.SaveChanges();

        //Without this line, no changes will get applied to the database
        transaction.Commit();
    }
    catch (Exception ex)
    {
        //There is no need to call transaction.Rollback() here so the transaction object
        //will go out of scope and disposing will roll back automatically
    }
}
```

Note that it may be a developers' convention to call `transaction.Rollback()` explicitly, because it makes the code more self-explanatory. Also, there may be less well-known query providers for Entity Framework out there that don't implement `IDisposable` correctly, which would also require an explicit `transaction.Rollback()` call.

Entity Framework Notes for Professionals

80+ pages
of professional hints and tricks

Contents

About	1
Chapter 1: Getting started with Entity Framework	2
Section 1.1: Installing the Entity Framework NuGet Package	2
Section 1.2: Using Entity Framework from C# (Code First)	4
Section 1.3: What is Entity Framework?	5
Chapter 2: Code First Conventions	6
Section 2.1: Removing Conventions	6
Section 2.2: Primary Key Convention	6
Section 2.3: Type Discovery	6
Section 2.4: DecimalPropertyConvention	7
Section 2.5: Relationship Convention	9
Section 2.6: Foreign Key Convention	10
Chapter 3: Code First DataAnnotations	11
Section 3.1: [Column] attribute	11
Section 3.2: [DatabaseGenerated] attribute	11
Section 3.3: [Required] attribute	12
Section 3.4: [MaxLength] and [MinLength] attributes	12
Section 3.5: [InverseProperty(string)] attribute	13
Section 3.6: [ComplexType] attribute	14
Section 3.7: [ForeignKey(string)] attribute	15
Section 3.8: [Range(min,max)] attribute	15
Section 3.9: [NotMapped] attribute	16
Section 3.10: [Table] attribute	17
Section 3.11: [Index] attribute	17
Section 3.12: [Key] attribute	18
Section 3.13: [StringLength(int)] attribute	19
Section 3.14: [Timestamp] attribute	19
Section 3.15: [ConcurrencyCheck] Attribute	20
Chapter 4: Entity Framework Code First	21
Section 4.1: Connect to an existing database	21
Chapter 5: Entity framework Code First Migrations	23
Section 5.1: Enable Migrations	23
Section 5.2: Add your first migration	23
Section 5.3: Doing "Update-Database" within your code	25
Section 5.4: Seeding Data during migrations	25
Section 5.5: Initial Entity Framework Code First Migration Step by Step	26
Section 5.6: Using Sql() during migrations	27
Chapter 6: Inheritance with EntityFramework (Code First)	29
Section 6.1: Table per hierarchy	29
Section 6.2: Table per type	29
Chapter 7: Code First - Fluent API	31
Section 7.1: Mapping models	31
Section 7.2: Composite Primary Key	32
Section 7.3: Maximum Length	33
Section 7.4: Primary Key	33
Section 7.5: Required properties (NOT NULL)	34

Section 7.6: Explicit Foreign Key naming	34
Chapter 8: Mapping relationship with Entity Framework Code First: One-to-one and variations	36
Section 8.1: Mapping one-to-zero or one	36
Section 8.2: Mapping one-to-one	39
Section 8.3: Mapping one or zero-to-one or zero	40
Chapter 9: Mapping relationship with Entity Framework Code First: One-to-many and Many-to-many	41
Section 9.1: Mapping one-to-many	41
Section 9.2: Mapping one-to-many: against the convention	42
Section 9.3: Mapping zero or one-to-many	43
Section 9.4: Many-to-many	44
Section 9.5: Many-to-many: customizing the join table	45
Section 9.6: Many-to-many: custom join entity	46
Chapter 10: Database first model generation	49
Section 10.1: Generating model from database	49
Section 10.2: Adding data annotations to the generated model	50
Chapter 11: Complex Types	52
Section 11.1: Code First Complex Types	52
Chapter 12: Database Initialisers	53
Section 12.1: CreateDatabaseIfNotExists	53
Section 12.2: DropCreateDatabaseIfModelChanges	53
Section 12.3: DropCreateDatabaseAlways	53
Section 12.4: Custom database initializer	53
Section 12.5: MigrateDatabaseToLatestVersion	54
Chapter 13: Tracking vs. No-Tracking	55
Section 13.1: No-tracking queries	55
Section 13.2: Tracking queries	55
Section 13.3: Tracking and projections	55
Chapter 14: Transactions	57
Section 14.1: Database.BeginTransaction()	57
Chapter 15: Managing entity state	58
Section 15.1: Setting state Added of a single entity	58
Section 15.2: Setting state Added of an object graph	58
Chapter 16: Loading related entities	60
Section 16.1: Eager loading	60
Section 16.2: Explicit loading	60
Section 16.3: Lazy loading	61
Section 16.4: Projection Queries	61
Chapter 17: Model Restraints	63
Section 17.1: One-to-many relationships	63
Chapter 18: Entity Framework with PostgreSQL	65
Section 18.1: Pre-Steps needed in order to use Entity Framework 6.1.3 with PostgresSql using NpgsqlDataProvider	65
Chapter 19: Entity Framework with SQLite	66
Section 19.1: Setting up a project to use Entity Framework with an SQLite provider	66
Chapter 20: .t4 templates in entity framework	69
Section 20.1: Dynamically adding Interfaces to model	69

End of ebook preview

Download the full PDF tutorial from the link below :

[Click Here](#)