Understanding Entity Framework Code First Migrations

*Posted By : Shailendra Chauhan, 18 Dec 2013*

*Updated On : 18 Sep 2014*

*Total Views : 30,623*

*Version Support : EF 6.0 & EF5.0*

*Keywords : entity framework code first migrations with example, code first migrations in entity framework*

Code-First approach allows you to define model classes as per the Domain requirements via POCOs. Hence, you have complete control over the classes being written or Implemented. Code First Migrations allow you to create a new database or to update existing database based on your model classes by using Package Manager Console exist within Visual Studio 2013 or Visual Studio 2012.

Code First Database Initializers

Before understanding code first migrations, let's have a look at Code First Database Initializers provided by Entity Framework. When you follow EF code first approach, then you have three options for initializing database as given below–

1. CreateDatabaseIfNotExists

This is the default database initializer class used by Code First. This class creates a database only if it doesn't exist. This initializer helps you to avoid any accidental deletion of the database.

1. DropCreateDatabaseWhenModelChanges

This database initializer class drop the existing database and re-creates it if there is a mismatch between the model classes and table schema. This initializer is useful during starting phase of development and testing when models are changing often and there is no concern about the existing database records.

1. DropCreateDatabaseAlways

This database initializer class always drop and creates a new database, whether it is already present or not with every run of the application. This initializer is useful during testing when you want to run the application with a fresh set of data.

Why Code First Migrations?

The above three database initializing approach become fail when you add new model classes and you want to update existing database without any deletion or change. To achieve this, you need to use EF code first migrations which allow you to update existing database with new model classes changes and your existing database remains same with your database records.

Note

You can also create custom database initializers by implementing the IDatabaseInitializer interface.

Code First Migrations in Entity Framework step by step

Model Classes

Suppose you have the following two entities within your application.

1. **public class Category**
2. **{**
3. **public int CategoryID { get; set; }**
4. **public string Name { get; set; }**
5. **}**
7. **public class Product**
8. **{**
9. **public int ProductID { get; set; }**
10. **public string Name { get; set; }**
11. **public decimal Price { get; set; }**
12. **public int CatID { get; set; }**
14. **public virtual Category Category { get; set; }**
15. **}**

Mapping Details

DataContext class and Fluent API mapping details for above two entities are given below –

1. **public class DataContext : DbContext**
2. **{**
3. **public DataContext():base("DefaultConnection"){ }**
5. **protected override void OnModelCreating(DbModelBuilder modelBuilder)**
6. **{**
7. **modelBuilder.Entity<Category>().HasKey(c => c.CategoryID);**
8. **modelBuilder.Entity<Category>().Property(p => p.Name).HasColumnType("VARCHAR").IsRequired().HasMaxLength(50);**
10. **modelBuilder.Entity<Product>().HasKey(p => p.ProductID);**
11. **modelBuilder.Entity<Product>().Property(p => p.Name).HasColumnType("VARCHAR").IsRequired().HasMaxLength(50);**
12. **modelBuilder.Entity<Product>().Property(p => p.Price).HasColumnType("DECIMAL").IsRequired();**
14. **modelBuilder.Entity<Product>().HasRequired(m => m.Category).WithMany().HasForeignKey(c => c.CatID);**
16. **}**
17. **public DbSet<Product> Product { get; set; }**
18. **public DbSet<Category> Category { get; set; }**
19. **}**

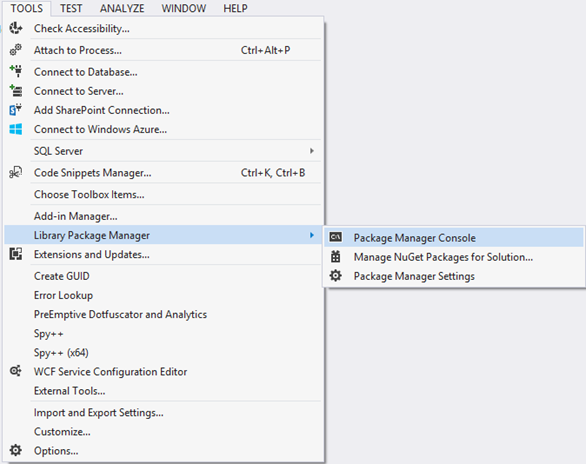
Connection String

And, your database connection string is given below:

1. **<connectionStrings>**
2. **<add name="DefaultConnection" providerName="System.Data.SqlClient" connectionString="data source=SHAILENDRA\SQLEXPRESS;initial catalog=EFCodeFirst;persist security info=True;user id=sa;password=mypassword; App=EntityFramework" /> </connectionStrings>**

Visual Studio Package Manager Console

To create the database for these two entities within your application, go to the Package Manager Console option as shown below:



Creating New Database

Running Commands

Run the following command to configure migrations within your project and for creating new database.

1. Enable migrations

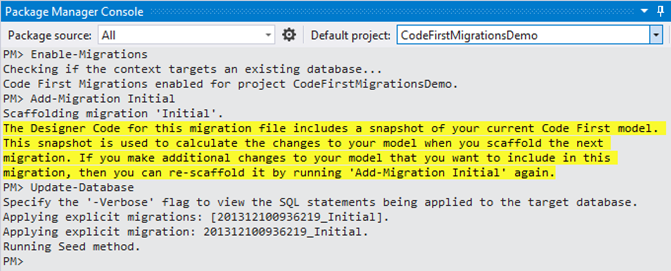
Enable-Migrations

1. Create migration

Add-Migration *MigrationsName*

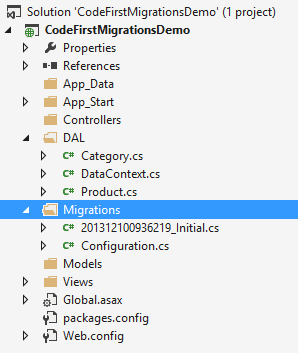
1. Create upgrade/downgrade script

Update-Database

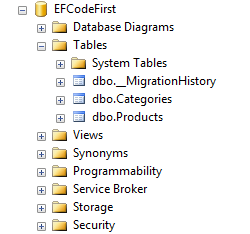


When you run above listed command on the Package Manager Console windows one by one then:

1. Firstly a Migrations folder will be added into your applications having Configuration.cs file for Migrations configuration setting.



1. Secondly a class file will be created having name suffix as MigrationsName followed by underscore and a unique generated number. This file will have all the entities to be created in the database.
2. Thirdly, a new database will be created having initial catalog name (initial catalog = YourDBName) as given in your connections string.



Updating Existing Database

Suppose you have added one more class named as Customer into your data model classes as given below:

Newly added Model Class

1. **public class Customer**
2. **{**
3. **public int CustomerID { get; set; }**
4. **public string Name { get; set; }**
5. **public string Address { get; set; }**
6. **}**

Running Commands

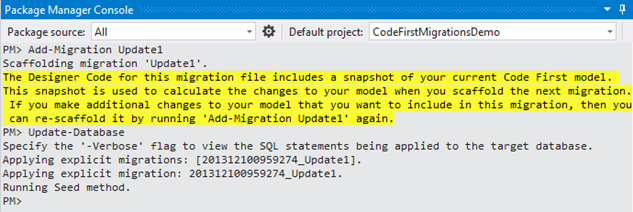
Now for updating the database with new changes run the following commands as given below:

1. Create migration

Add-Migration *MigrationsName*

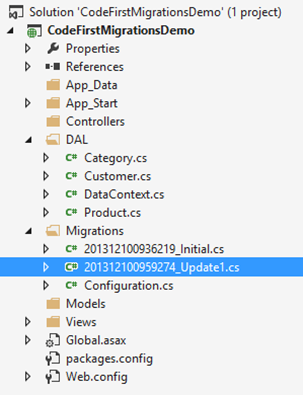
1. Create upgrade/downgrade script

Update-Database

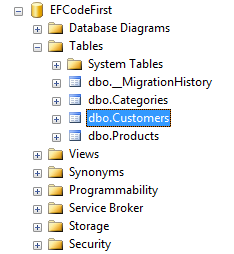


When you run above listed command on the Package Manager Console windows one by one then

1. Firstly a class file will be created having name suffix as MigrationsName followed by underscore and a unique generated number. This file will have all the entities to be created or updated in the database.



1. Secondly, the existing database will be updated with new changes.



Undo/Rollback a Migrations

You can also Undo/Rollback a specific migrations by using following commands:

1. Rollback to a specific migrations

Update-Database -TargetMigration:*MigrationsName*

1. Rollback all migrations

Update-Database -TargetMigration:0

**Some useful Migrations power shell scripts**

1. Enable-Migrations
2. Add-Migration Department (Department is class name)
3. Update-Database (To update if any model changes)
4. Add-Migration RemoveCourse (To remove any coloum “Course”)
5. Add-Migration RefIne (Adding reference PK and FK to existing Database)

