Features for Database First (Designer) and Code-First Both:

* Connection resiliency
* Asynchronous query and save
* Code-based configuration
* Database command logging
* Database command interception
* Dependency Resolution
* DbSet.AddRange/RemoveRange
* Better Transaction Support
* Pluggable pluralisation and singularization service
* Testability improvements
* Creating context with an open connection
* Improved performance and warm-up time

Features for Code-First:

* Custom conventions
* Insert, update & delete stored procedures for entity CUD operation
* Index attribute (EF 6.1)
* Multiple context per database
* Nested entity types
* Custom migration operations
* Configurable migration history table

## **Asynchronous Query:**

private static async Task<Student> GetStudent()

{

Student myStudent = null;

using (var context = new SchoolDBEntities())

{

Console.WriteLine("Start GetStudent...");

myStudent = await (context.Students.Where(s => s.StudentID == 1).FirstOrDefaultAsync<Student>());

Console.WriteLine("Finished GetStudent...");

}

return student;

}

## **Asynchronous Save:**

You can call context.SaveChanges asynchronously the same way as async query:

private static async Task SaveStudent(Student editedStudent)

{

using (var context = new SchoolDBEntities())

{

context.Entry(editedStudent).State = EntityState.Modified;

Console.WriteLine("Start SaveStudent...");

int x = await (context.SaveChangesAsync());

Console.WriteLine("Finished SaveStudent...");

}

}

public static void AsyncQueryAndSave()

{

var queryResult = GetStudent();

Console.WriteLine("Let's do something else till we get student..");

queryResult.Wait();

var student1 = queryResult.Result;

student1.FirstName = "Modified First Name";

var studentSave = SaveStudent(student1);

Console.WriteLine("Let's do something else till we save student.." );

studentSave.Wait();

}

# **Code-based Configuration:**

public class FE6CodeConfig : DbConfiguration

{

public FE6CodeConfig()

{

//define configuration here

}

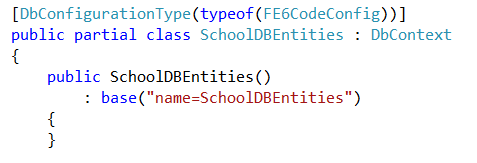
}

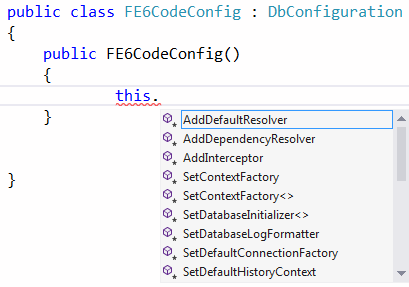
}

<entityFramework codeConfigurationType="EF6DBFirstTutorials.FE6CodeConfig, EF6DBFirstTutorials">

</entityFramework>

Or





### Set default connection factory:

Code-based configuration:

public class FE6CodeConfig : DbConfiguration

{

public FE6CodeConfig()

{

this.SetDefaultConnectionFactory(new System.Data.Entity.Infrastructure.SqlConnectionFactory());

}

}

config file:

<entityFramework>

<defaultConnectionFactory type="System.Data.Entity.Infrastructure.SqlConnectionFactory, EntityFramework" />

</entityFramework>

### Set Database Provider:

Code-based configuration:

public class FE6CodeConfig : DbConfiguration

{

public FE6CodeConfig()

{

this.SetProviderServices("System.Data.SqlClient",

System.Data.Entity.SqlServer.SqlProviderServices.Instance);

}

}

config file:

<entityFramework>

<providers>

<provider invariantName="System.Data.SqlClient" type="System.Data.Entity.SqlServer.SqlProviderServices, EntityFramework.SqlServer" />

</providers>

</entityFramework>

## **Set Database Initializers:**

You can set database initializers (for Code-First only) using code-based configuration as shown below:

public class FE6CodeConfig : DbConfiguration

{

public FE6CodeConfig()

{

this.SetDatabaseInitializer<SchoolDBEntities>(new CustomDBInitializer<SchoolDBEntities>());

}

}

config file:

<entityFramework>

<contexts>

<context type="EF6DBFirstTutorials.SchoolDBEntities, EF6DBFirstTutorials" >

<databaseInitializer type="EF6DBFirstTutorials.CustomDBInitializer , EF6DBFirstTutorials">

</databaseInitializer>

</context>

</contexts>

</entityFramework>

public class Logger

{

public static void Log(string message)

{

Console.WriteLine("EF Message: {0} ", message);

}

}

# **Database Command Logging:**

class EF6Demo

{

public static void DBCommandLogging()

{

using (var context = new SchoolDBEntities())

{

context.Database.Log = Logger.Log;

var student = context.Students

.Where(s => s.StudentName == "Student1").FirstOrDefault<Student>();

student.StudentName = "Edited Name";

context.SaveChanges();

}

}

}

class EFCommandInterceptor: IDbCommandInterceptor

{

public void NonQueryExecuted(System.Data.Common.DbCommand command, DbCommandInterceptionContext<int> interceptionContext)

{

LogInfo("NonQueryExecuted", String.Format(" IsAsync: {0}, Command Text: {1}", interceptionContext.IsAsync, command.CommandText));

}

public void NonQueryExecuting(System.Data.Common.DbCommand command, DbCommandInterceptionContext<int> interceptionContext)

{

LogInfo("NonQueryExecuting", String.Format(" IsAsync: {0}, Command Text: {1}", interceptionContext.IsAsync, command.CommandText));

}

public void ReaderExecuted(System.Data.Common.DbCommand command, DbCommandInterceptionContextt<System.Data.Common.DbDataReader> interceptionContext)

{

LogInfo("ReaderExecuted", String.Format(" IsAsync: {0}, Command Text: {1}", interceptionContext.IsAsync, command.CommandText));

}

public void ReaderExecuting(System.Data.Common.DbCommand command, DbCommandInterceptionContext<System.Data.Common.DbDataReader> interceptionContext)

{

LogInfo("ReaderExecuting", String.Format(" IsAsync: {0}, Command Text: {1}", interceptionContext.IsAsync, command.CommandText));

}

public void ScalarExecuted(System.Data.Common.DbCommand command, DbCommandInterceptionContext<object> interceptionContext)

{

LogInfo("ScalarExecuted", String.Format(" IsAsync: {0}, Command Text: {1}", interceptionContext.IsAsync, command.CommandText));

}

public void ScalarExecuting(System.Data.Common.DbCommand command, DbCommandInterceptionContext<object> interceptionContext)

{

LogInfo("ScalarExecuting", String.Format(" IsAsync: {0}, Command Text: {1}", interceptionContext.IsAsync, command.CommandText));

}

private void LogInfo(string command, string commandText)

{

Console.WriteLine("Intercepted on: {0} :- {1} ", command, commandText);

}

}

You can see in the above code that **IDbCommandInterceptor** provides six methods to execute.

Now, you will need to configure the interceptor either by using config file or code-based configuration.

Config file:

<entityFramework>

<interceptors>

<interceptor type="EF6DBFirstTutorials.EFCommandInterceptor, EF6DBFirstTutorials">

</interceptor>

</interceptors>

</entityFramework>

Code-based config:

public class FE6CodeConfig : DbConfiguration

{

public FE6CodeConfig()

{

this.AddInterceptor(new EFCommandInterceptor());

}

}

# **Transaction support:**

using (System.Data.Entity.DbContextTransaction dbTran = context.Database.BeginTransaction( ))

{

try

{

Student std1 = new Student() { StudentName = "newstudent" };

context.Students.Add(std1);

context.Database.ExecuteSqlCommand(

@"UPDATE Student SET StudentName = 'Edited Student Name'" +

" WHERE StudentID =1"

);

context.Students.Remove(std1);

//saves all above operations within one transaction

context.SaveChanges();

//commit transaction

dbTran.Commit();

}

catch (Exception ex)

{

//Rollback transaction if exception occurs

dbTran.Rollback();

}

}

# **DbSet.AddRange & DbSet.RemoveRange:**

IList<Student> newStudents = new List<Student>();

newStudents.Add(new Student() { StudentName = "Student1 by addrange" });

newStudents.Add(new Student() { StudentName = "Student2 by addrange" });

newStudents.Add(new Student() { StudentName = "Student3 by addrange" });

using (var context = new SchoolDBEntities())

{

context.Students.AddRange(newStudents);

context.SaveChanges();

}

Similarly, DbSet.RemoveRange is used to remove collection of entities from DbSet.

IList<Student> existingStudents = …..

using (var context = new SchoolDBEntities())

{

context.Students.RemoveRange(existingStudents);

context.SaveChanges();

}

# **Code First - Insert, Update, Delete Stored Procedure Mapping:**

class Student

{

public Student()

{

}

public int Student\_ID { get; set; }

public string StudentName { get; set; }

}

protected override void OnModelCreating(DbModelBuilder modelBuilder)

{

modelBuilder.Entity<Student>()

.MapToStoredProcedures();

}

protected override void OnModelCreating(DbModelBuilder modelBuilder)

{

modelBuilder.Entity<Student>()

.MapToStoredProcedures(p => p.Insert(sp => sp.HasName("sp\_InsertStudent").Parameter(pm => pm.StudentName, "name").Result(rs => rs.Student\_ID, "Student\_ID"))

.Update(sp => sp.HasName("sp\_UpdateStudent").Parameter(pm => pm.StudentName, "name"))

.Delete(sp => sp.HasName("sp\_DeleteStudent").Parameter(pm => pm.Student\_ID, "Id"))

);

}

protected override void OnModelCreating(DbModelBuilder modelBuilder)

{

modelBuilder.Types().Configure(t => t.MapToStoredProcedures());

}

# **Index Attribute:**

class Student

{

public Student()

{

}

public int Student\_ID { get; set; }

public string StudentName { get; set; }

[Index]

public int RegistrationNumber { get; set; }

}

Coding Convention

After table creation change column length

public class NorthwindContext:DbContext  
{  
    public DbSet<Customer> Customers { get; set; }  
    public DbSet<Employee> Employees { get; set; }  
    public DbSet<Product> Products { get; set; }  
    public DbSet<Order> Orders { get; set; }

    protected override void OnModelCreating(DbModelBuilder modelBuilder)  
    {  
        modelBuilder.Properties<string>().Configure(col => col.HasMaxLength(300));  
        modelBuilder.Properties().Where(p => p.Name == "ProductName").Configure(p => p.HasColumnName("Name"));  
    }  
}