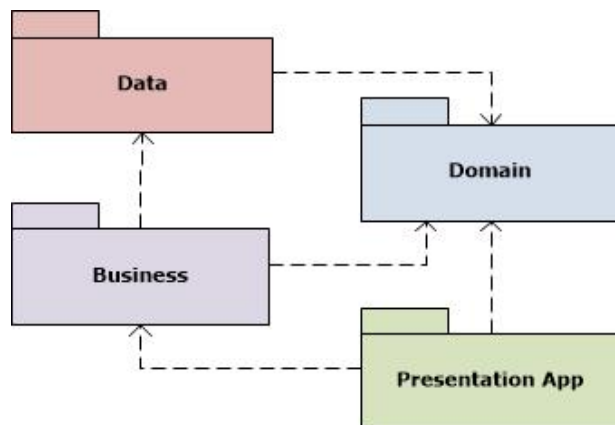


Introduction

This document will provide a step by step work flow of the Entity Framework Core Code First approach. This approach follows a domain-driven design that requires the domain model (business logic layer) to be created first. This document is directed to users of ASP.NET Core using Entity Framework Core

The architecture of the solution is broken down into four projects. The image below shows the projects with their dependencies (e.g. Business project adds a reference to both Data and Domain projects):

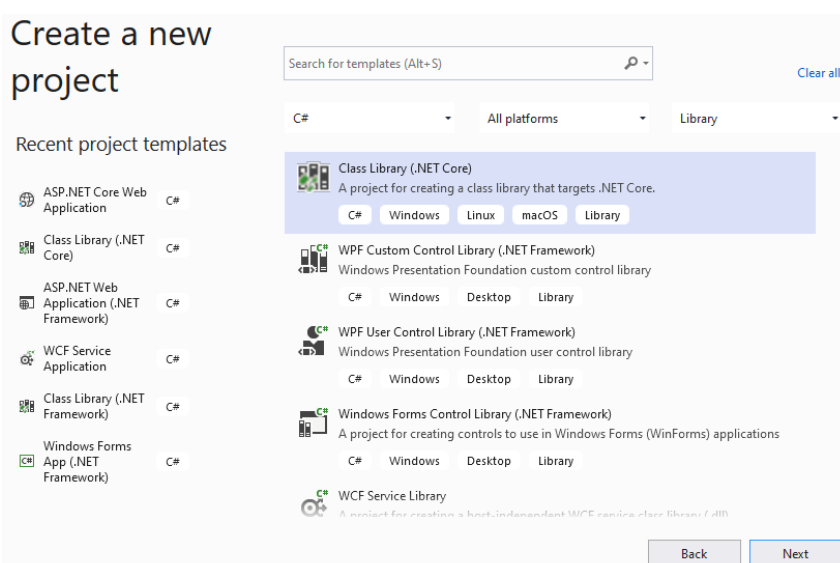


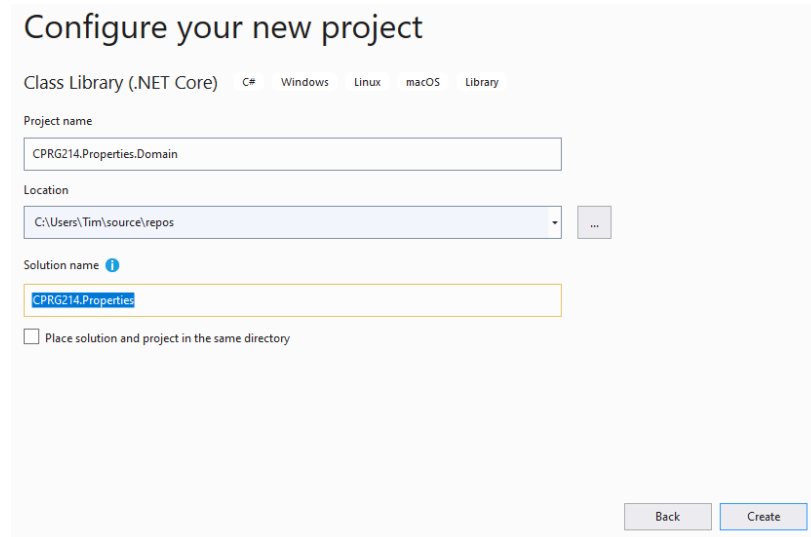
The Domain Project

In this demo, we will work with the property rentals domain and four classes: RentalProperty, PropertyType, Owner, and Renter.

Step 1: Create the domain library as a **Class Library (.NET Core)** project called CPRG214.Properties.Domain. I've changed the solution name to CPRG214.Properties only (this is the first project being created in the solution).

Create a new project





Step 2: Add a class to the domain project for each of the four domain classes. Ensure each class is public. The convention is one class per file.

Step 3: Code properties for the four classes based on the relationships that exist between the classes and if a property represents a required field. By convention, reference types are nullable and value types are not null by default. So, string properties of a domain entity would automatically be made nullable so the string properties require a data annotation to identify them as required fields. The Rent property is not null because decimal type is a value type. If a value type is going to be not null in the database then the data type has to be nullable. See the RenterId property in the RentalProperty class. The data type is int? which is a nullable int.

A RentalProperty has an owner, is rented by a renter and is of a designated property type. An owner can own many properties. A renter can rent only one property. PropertyType can be associated with many properties.

- Import the namespaces **System.ComponentModel.DataAnnotations** and **System.ComponentModel.DataAnnotations.Schema** to the files for each of the four classes.
- Code the navigation properties (properties that implement the one-to-many or one to one relationship such as RentalProperty and Renter).
- Add the [Required] attribute to each of the string properties that are mandatory.

```
[Table("RentalProperty")]
public class RentalProperty
{
    public int Id { get; set; }
    [Required]
    public string Address { get; set; }
    [Required]
    public string City { get; set; }
    [Required]
```

```
        public string Province { get; set; }
        [Required]
        public string PostalCode { get; set; }
        public decimal Rent { get; set; }
        public int PropertyTypeId { get; set; }
        public int OwnerId { get; set; }
        //nullable int required if FK is nullable
        public int? RenterId { get; set; }
        //navigation properties
        public PropertyType PropertyType { get; set; }
        public Owner Owner { get; set; }
        public Renter Renter { get; set; }
    }

    [Table("PropertyType")]
    public class PropertyType
    {
        public int Id { get; set; }
        [Required]
        public string Style { get; set; }
        //navigation property
        public ICollection<RentalProperty> RentalProperties { get; set; }
    }

    [Table("Owner")]
    public class Owner
    {
        public int Id { get; set; }
        [Required][Display(Name="Owner")]
        public string Name { get; set; }
        [Required]
        public string Phone { get; set; }
        //navigation property
        public ICollection<RentalProperty> RentalProperties { get; set; }
    }

    [Table("Renter")]
    public class Renter
    {
        public int Id { get; set; }
        [Required]
        public string FirstName { get; set; }
        [Required]
        public string LastName { get; set; }
        [Required]
        public string Phone { get; set; }
        //navigation property (one-to-one)
        public RentalProperty RentalProperty { get; set; }
    }
```

The Data Project

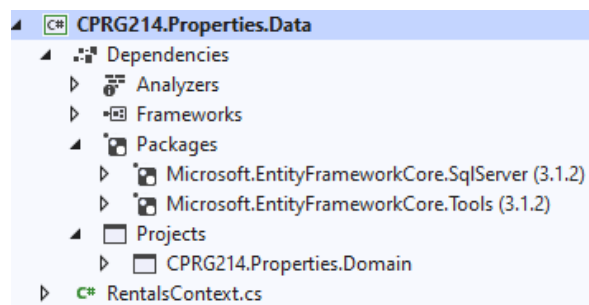
The data project will contain the Entity Framework reference and the context class that will be used to create the database based on the DbSet types defined in the Domain project. The data project has a dependency on the domain project so a reference needs to be added.

Step 4: Add the data project called CPRG214.Properties.Data as a .NET Core Class Library.

Step 5: Add a project reference to domain library.

Step 6: Add the Entity Framework Core dependency for the data provider to the Data project using the Nuget Package Manager. Browse for “Microsoft.EntityFrameworkCore.SqlServer” and select the package version for your .NET Core installation (v 3.1.2 for me currently) and click Install. Click the OK button on the preview dialog and then accept the license.

Step 7: Add the Entity Framework Core Tools package called Microsoft.EntityFrameworkCore.Tools to the project. You should see the two packages under the Dependencies folder of the Data project:



Step 8: Add a public class to this project called RentalsContext. This class inherits the base context class called DbContext. A DbSet collection property is added for each entity type in the domain model.

Two methods are overridden:

1. OnConfiguring has a DbContextOptionsBuilder instance use SQL Server as the provider and the connection string is passes in here.
2. OnModelCreating has a ModelBuilder instance that can be used to create seed data when the database is created.

This is different behavior from EF 6.2. Table names are pluralized by default. In order to not pluralize table names requires a data annotation.

```
using CPRG102.Properties.Domain;
using Microsoft.EntityFrameworkCore;
using System;

namespace CPRG214.Properties.Data
{
    public class RentalsContext : DbContext
    {
        public RentalsContext() : base() { }
    }
}
```

```

public DbSet<RentalProperty> RentalProperties { get; set; }
public DbSet<Owner> Owners { get; set; }
public DbSet<Renter> Renters { get; set; }
public DbSet<PropertyType> PropertyTypes { get; set; }

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
{
    //Change the connection string here for your home computer/lab computer
    optionsBuilder.UseSqlServer(@"Server=localhost;
                                Database=PropertyRentals;
                                Trusted_Connection=True;");
}

protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    base.OnModelCreating(modelBuilder);

    //seed data created here
    modelBuilder.Entity<PropertyType>().HasData(
        new PropertyType { Id = 1, Style = "Apartment"},
        new PropertyType { Id = 2, Style = "Townhouse" },
        new PropertyType { Id = 3, Style = "Bungalow" },
        new PropertyType { Id = 4, Style = "Suite" }
    );

    modelBuilder.Entity<Owner>().HasData(
        new Owner
        {
            Id = 1,
            Name = "Ben King",
            Phone = "403-555-8500"
        },
        new Owner
        {
            Id = 2,
            Name = "Paul Wilson",
            Phone = "403-555-6935"
        },
        new Owner
        {
            Id = 3,
            Name = "Kimberly Dunne",
            Phone = "403-555-4770"
        }
    );

    modelBuilder.Entity<Renter>().HasData(
        new Renter
        {
            Id = 1,
            FirstName = "Sam",
            LastName = "Munro",
            Phone = "403-555-3456"
        },
        new Renter
        {
            Id = 2,

```

```

        FirstName = "Sarah",
        LastName = "Carr",
        Phone = "403-555-7666"
    },
    new Renter
    {
        Id = 3,
        FirstName = "John",
        LastName = "Hudon",
        Phone = "403-555-3000"
    }
    );

modelBuilder.Entity<RentalProperty>().HasData(
    new RentalProperty
    {
        Id = 1,
        Address = "1345 - 670 14th Ave SW",
        City = "Calgary",
        Province = "AB",
        PostalCode = "T3T 3T3",
        Rent = 1200m,
        PropertyTypeId = 1,
        OwnerId = 2,
        RenterId = 3
    },
    new RentalProperty
    {
        Id = 2,
        Address = "4567 66th Ave NW",
        City = "Calgary",
        Province = "AB",
        PostalCode = "T2T 2D2",
        Rent = 2400m,
        PropertyTypeId = 3,
        OwnerId = 1,
        RenterId = 2
    },
    new RentalProperty
    {
        Id = 3,
        Address = "240 - 2111 4th St NW",
        City = "Calgary",
        Province = "AB",
        PostalCode = "T5T 5T5",
        Rent = 1000m,
        PropertyTypeId = 4,
        OwnerId = 3
    }
    );
}
}
}
}
}

```

The context class represents a session with the database that allows all the CRUD functionality, such as querying data, adding and updating data and saving changes.

The main namespace is Microsoft.EntityFrameworkCore.Data and this namespace needs to be imported to allow use of DbContext and DbSet types.

Step 9: Create the database and seed data by running code-based migrations using the Package Manager Console in Visual Studio. Ensure the Data project is selected as the “Default Project” and you might need to set it as the startup project as well.

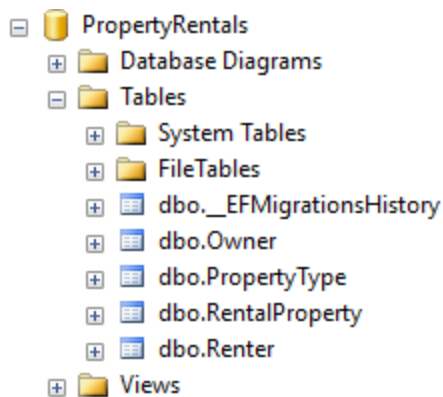
- a. Run Add-Migration command and provide a name such as CreateRentalProperties. This creates a class with Up() and Down() methods for creating the tables in the database.

```
PM> Add-Migration CreateRentalProperties
Build started...
Build succeeded.
To undo this action, use Remove-Migration.
```

- b. Run the Update-Database command

```
PM> Update-Database
Build started...
Build succeeded.
Applying migration '20200414030108_CreateRentalProperties'.
Done.
PM> |
```

The database will be called PropertyRentals (set in the connection string) and contain the four tables of the EDM plus another table called _EFMigrationsHistory that will track the migrations (changes) to the database starting with the “CreateProperties” migration that created the four tables.



The RentalProperty table has the following result set:

PropertyRentals.dbo.RentalProperty								
Id	Address	City	Province	PostalCode	Rent	PropertyTypeId	OwnerId	RenterId
1	1345 - 670 14th ...	Calgary	AB	T3T 3T3	1200.00	1	2	3
2	4567 66th Ave ...	Calgary	AB	T2T 2D2	2400.00	3	1	2
3	240 - 2111 4th S...	Calgary	AB	T5T 5T5	1000.00	4	3	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

The Business Logic Layer Project

The manager classes will be coded in this layer. The manager classes are defined in a class library separate from the domain entities because the Data project has a dependency on the Domain project. Managers reference the data context class so a dependency also exists there. The Domain project cannot have a dependency on the Data project as that would result in a circular reference which is not allowed.

Step 10: Add a class library called CPRG214.Properties.BLL.

Step 11: Add references to the Data and Domain projects.

Step 12: The project also requires a reference to the Entity Framework so the managers can work it. The NuGet Package Manager can be used by going to installed packages and selecting EF Core package.

Step 13: Add a Manager class for Rentals.

```
public class RentalsManager {
    public static List<RentalProperty> GetAll()
    {
        var context = new RentalsContext();
        var rentals = context.RentalProperties.Include(r => r.Owner).ToList();
        return rentals;
    }
}
```

This is not a complete list of the methods required but enough to develop a page displaying a list of rental properties. Note the call to the Include method so Owner records are loaded.

PRESENTATION APP PROJECT

Step 12. Add an ASP.NET Core Web Application (Razor Page) project to the solution that has dependencies on the Business and Domain projects. Call the project CPRG214.Properties.App.

Step 13. Add page called Rentals to the Pages folder.

Step 14: Import Domain and BLL namespaces.

Step 15: Modify code:

```
5 references
public class RentalsModel : PageModel
{
    5 references
    public IList<RentalProperty> RentalProperties { get; set; }
    0 references
    public void OnGet()
    {
        RentalProperties = PropertiesManager.GetAll();
    }
}
```


Step 16: Modify the markup file:

```
@page
@model CPRG214.Properties.App.Pages.RentalsModel
@{
    ViewData["Title"] = "Rentals";
}

<h1>Rental Properties List</h1>

<table class="table">
<thead>
<tr>
<th>
    @Html.DisplayNameFor(model => model.RentalProperties[0].Address)
</th>
<th>
    @Html.DisplayNameFor(model => model.RentalProperties[0].City)
</th>
<th>
    @Html.DisplayNameFor(model => model.RentalProperties[0].Owner.Name)
</th>
</tr>
</thead>
<tbody>
    @foreach (var item in Model.RentalProperties)
    {
        <tr>
        <td>
            @Html.DisplayFor(modelItem => item.Address)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.City)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Owner.Name)
        </td>
        </tr>
    }
</tbody>
</table>
```

Run and test the application.

