



How to implement Repository Design Pattern in C#

Back to: [Dot Net Design Patterns With Real-Time Examples](#)

Simple Dress Patterns

Women's Dress Patterns

Employee Satisfaction Survey Questions

Free Jelly Roll Quilt Pattern

Easy Hobo Bag Patterns

Employee Performance Evaluation

How to Implement Repository Design Pattern in C# with Real-time Examples

In this article, I am going to discuss **How to Implement Repository Design Pattern in C#** using ASP.NET MVC [application](#) with the Entity Framework. I strongly recommended you to read our previous article before proceeding to this article where we discussed the [basics of Repository Design Pattern](#) in C#.

As we already discussed in our previous article that the Repository Design Pattern is used to create an abstraction layer between the data access layer and the business logic layer of the application. That abstraction layer is generally called as the Repository Layer and it will directly communicate with the data access layer, gets the data and provides it to the business logic layer.

The main advantage to use the repository design pattern is to isolate the data access logic and business logic. So that if we do any changes in any of this logic, then that should affect other logic. So let us discuss the step by step procedure to implement the repository design pattern in C#.

STEP1: Create the Required Database tables

We are going to use the following Employee table in this demo.

EmployeeID	Name	Gender	Salary	Dept
1	Pranaya	Male	10000	IT
2	Anurag	Male	15000	HR
3	Priyanka	Female	22000	HR
4	Sambit	Male	20000	IT
5	Preety	Female	25000	IT
6	Hina	Female	20000	HR

Please use the below SQL script to create and populate the Employee table with the required data that we are going to use in our application.

```
-- Create EmployeeDB database
```

```
CREATE DATABASE EmployeeDB
```

```
GO
```

```
USE EmployeeDB
```

```
GO
```

```
-- Create Employee Table
```

```
CREATE TABLE Employee
```

```
(
```

```
    EmployeeID INT PRIMARY KEY IDENTITY(1,1),
```

```
    Name VARCHAR(100),
```

```
    Gender VARCHAR(100),
```

```
    Salary INT,
```

```
    Dept VARCHAR(50)
```

```
)
```

```
GO
```

```
-- Populate some test data into Employee table
```

```
INSERT INTO Employee VALUES('Pranaya', 'Male', 10000, 'IT' )
```

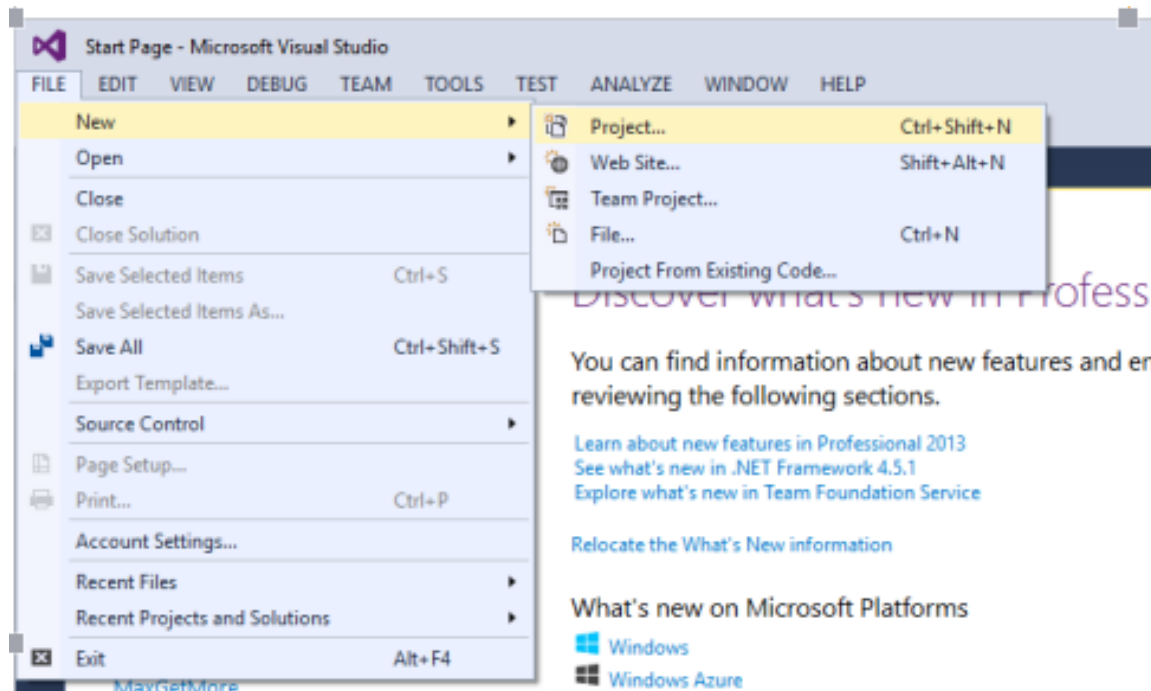
```
INSERT INTO Employee VALUES('Anurag', 'Male', 15000, 'HR' )
```

```
INSERT INTO Employee VALUES('Priyanka', 'Female', 22000, 'HR' )
INSERT INTO Employee VALUES('Sambit', 'Male', 20000, 'IT' )
INSERT INTO Employee VALUES('Preety', 'Female', 25000, 'IT' )
INSERT INTO Employee VALUES('Hina', 'Female', 20000, 'HR' )
GO
```

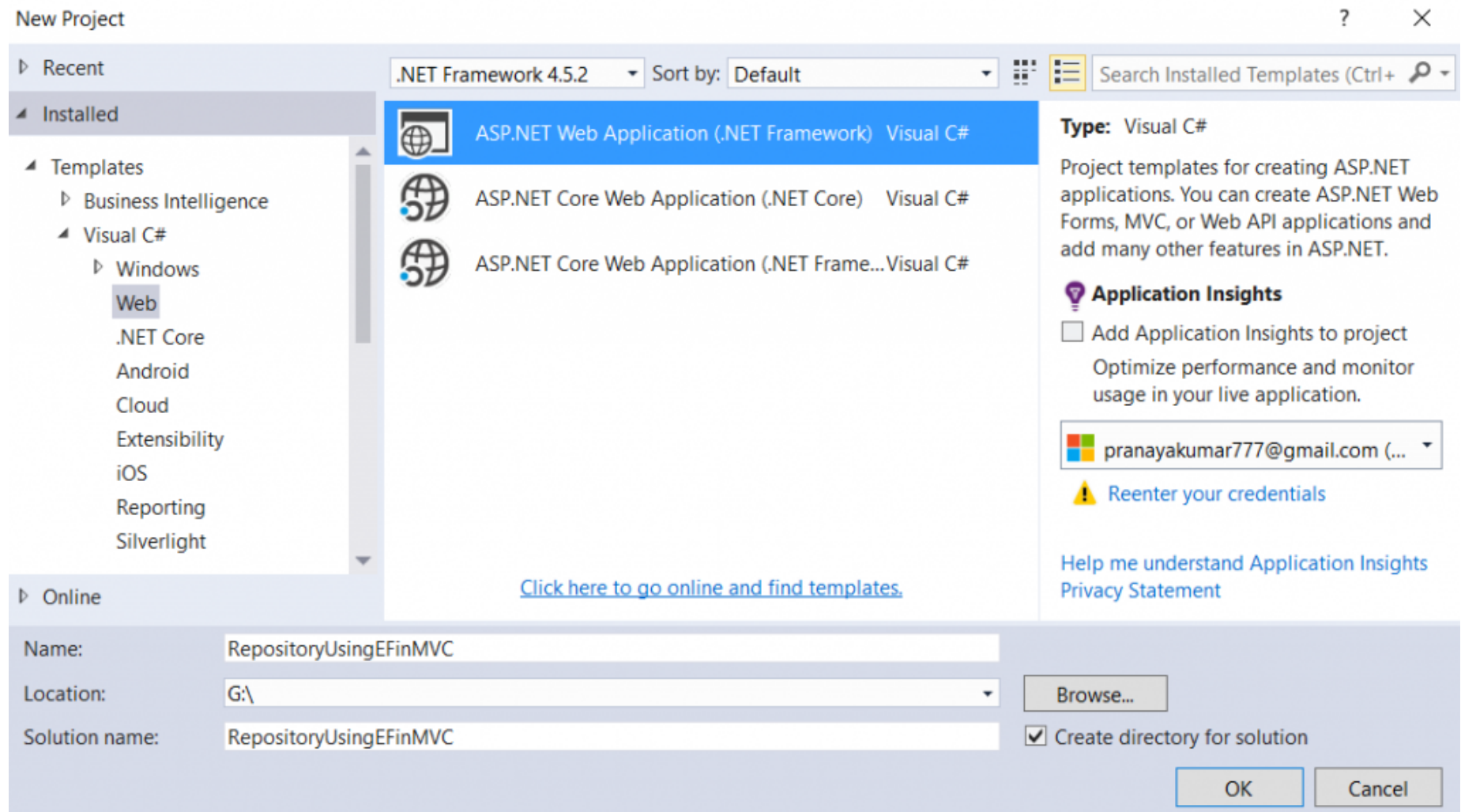
```
SELECT * FROM Employee
GO
```

STEP2: Create a new ASP.NET MVC application

Open Visual Studio and create a new project. To do so, Select **File => New => Project** option as shown in the below image.



After clicking on the “**Project**” link a new dialog will pop up. In that, we are going to select web templates from the left pane. From the middle pane, we need to select “**ASP.NET Web Application**”. Provide a meaningful name the project such as “**RepositoryUsingEFinMVC**” and then click on the OK [button](#) as shown in the below image.



Once you click on the OK button, it will open a new dialog pop up with Name “New ASP.NET Project” for selecting project Templates. Here, we are going to choose the MVC project template. Then we are going to choose the Authentication type for our application. For selecting the authentication, just click on Change Authentication button, a new dialog will pop up with the name “Change Authentication” and from there we are going to choose No Authentication and then click on the OK Button as shown below.

New ASP.NET Web Application - RepositoryUsingEFInMVC



Select a template:

ASP.NET 4.5.2 Templates

Empty



Web Forms



MVC



Web API



Single Page Application



Azure API App



Azure Mobile Service

A project template for creating ASP.NET MVC applications. ASP.NET MVC allows you to build applications using the Model-View-Controller architecture. ASP.NET MVC includes many features that enable fast, test-driven development for creating applications that use the latest standards.

[Learn more](#)[Change Authentication](#)Authentication: **No Authentication** **Microsoft Azure** ☐ Host in the cloud

App Service

Add folders and core references for:

☐ Web Forms ☒ MVC ☐ Web API☐ Add unit testsTest project name:

OK

Cancel

Once you click on the OK button, it will take some time to create the project for us. Once the project is created let's see the folder structure as shown below.

Once the project is created next we need to add ADO.NET Entity Data Model

STEP3: Adding ADO.NET Entity Data Model

First, add a folder with the name DAL to our project.

To do so, right click on the Project => Add => New Folder

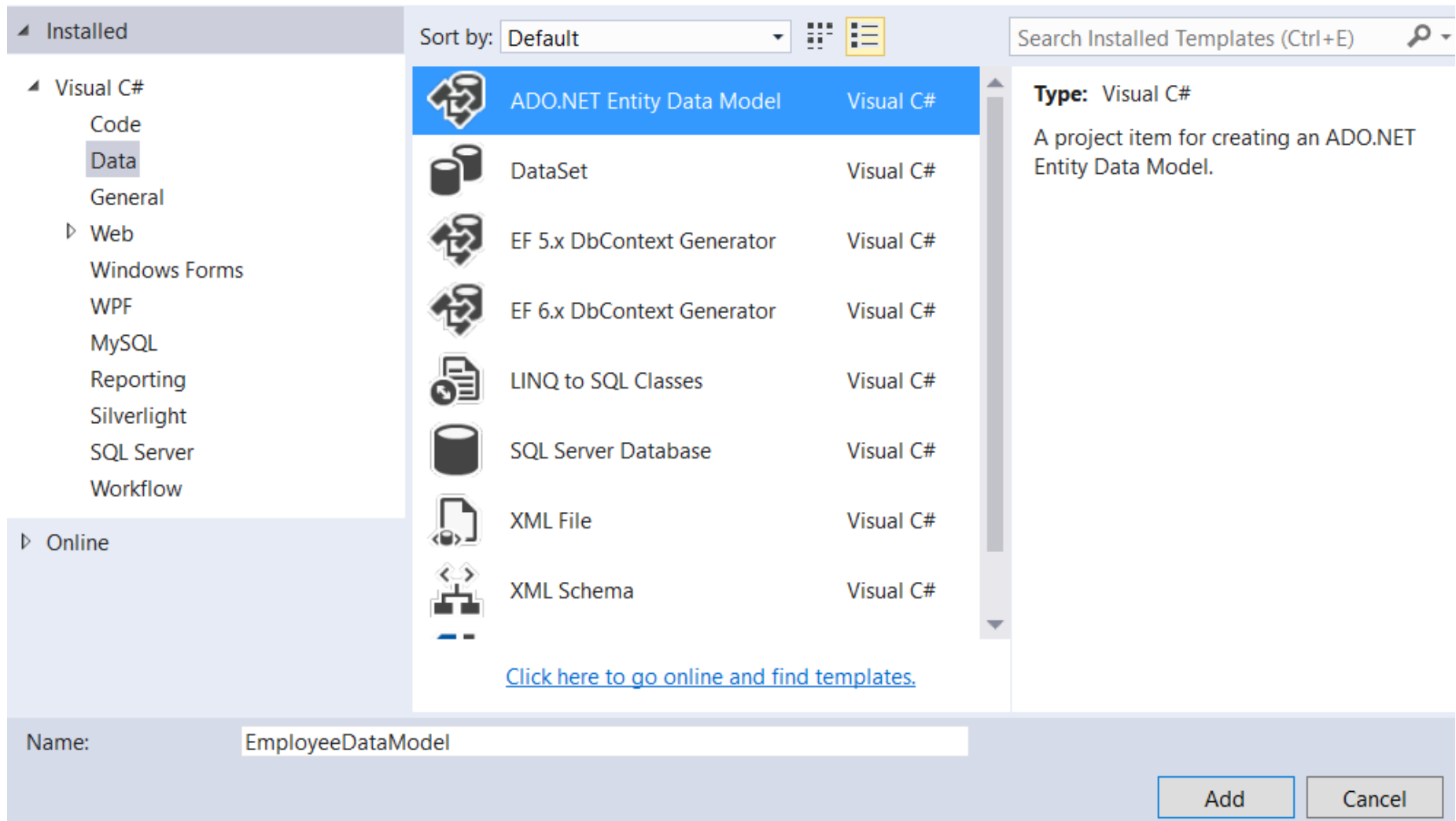
Rename the folder name as DAL

Next, add ADO.NET Entity Data Model inside DAL Folder

To do so, right-click on DAL folder then Add => New Item

Select ADO.NET Entity Data Model, Provide a meaningful name "EmployeeDataModel" and click on ADD button as shown below

Add New Item - RepositoryUsingEFinMVC



Next From the Choose Model Content Screen choose **"Generate From Database"** and Click Next button as shown below

Entity Data Model Wizard

**Choose Model Contents****What should the model contain?**

EF Designer
from
database



Empty EF
Designer
model



Empty Code
First model



Code First
from
database

Creates a model in the EF Designer based on an existing database. You can choose the database connection, settings for the model, and database objects to include in the model. The classes your application will interact with are generated from the model.

< Previous

Next >

Finish

Cancel

From the next screen Click on New Connection and provide the necessary details, Select the database and click on OK as shown below

Connection Properties



Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.

Data source:

Microsoft SQL Server (SqlClient)

Change...

Server name:

LAPTOP-2HN3PT8T\SQLEXPRESS

Refresh

Log on to the server

Authentication:

Windows Authentication

User name:

Password:

☐ Save my password

Connect to a database

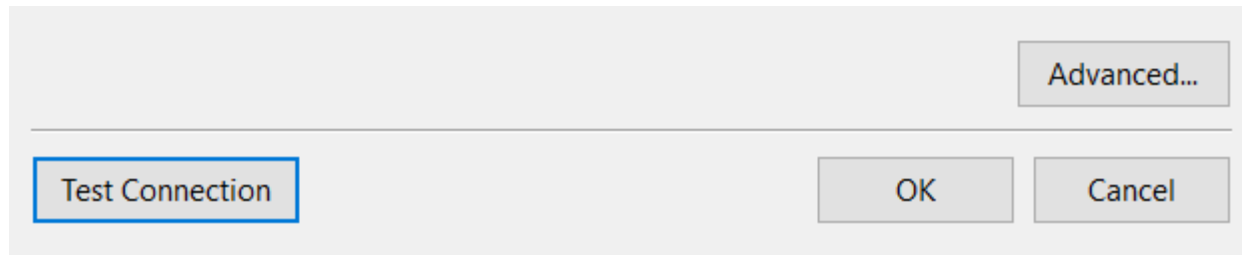
☒ Select or enter a database name:

EmployeeDB

☐ Attach a database file:

Browse...

Logical name:



In the next step provide a meaningful name "EmployeeDBContext" for the Connection String that is going to create in Web.config file and click on Next as shown below

Entity Data Model Wizard



Choose Your Data Connection

Which data connection should your application use to connect to the database?

laptop-2hn3pt8t\sqlexpress.EmployeeDB.dbo

New Connection...

This connection string appears to contain sensitive data (for example, a password) that is required to connect to the database. Storing sensitive data in the connection string can be a security risk. Do you want to include this sensitive data in the connection string?

- ☐ No, exclude sensitive data from the connection string. I will set it in my application code.
- ☐ Yes, include the sensitive data in the connection string.

Connection string:

```
metadata=res://*/DAL.EmployeeDataModel.csdl|res://*/DAL.EmployeeDataModel.ssdl|
res://*/DAL.EmployeeDataModel.msl;provider=System.Data.SqlClient;provider connection
string="data source=LAPTOP-2HN3PT8T\SQLEXPRESS;initial catalog=EmployeeDB;integrated
security=True;MultipleActiveResultSets=True;App=EntityFramework"
```

☒ Save connection settings in Web.Config as:

EmployeeDBContext

< Previous

Next >

Finish

Cancel

From Choose your version screen, choose Entity Framework 6.x and click on Next button as shown below

Entity Data Model Wizard

**Choose Your Version****Which version of Entity Framework do you want to use?**

☒ Entity Framework 6.x

☐ Entity Framework 5.0



It is also possible to install and use other versions of Entity Framework.

[Learn more about this](#)

< Previous

Next >






Finish

Cancel

In the next step, from Choose your database objects screen, choose the Employee object, provide the namespace name and click on Finish button as shown below

Entity Data Model Wizard

**Choose Your Database Objects and Settings****Which database objects do you want to include in your model?**

- ☒  **Tables**
- ☒  **dbo**
- ☒  **Employee**
- ☐  **Views**
- ☐  **Stored Procedures and Functions**

- ☒ Pluralize or singularize generated object names
- ☒ Include foreign key columns in the model
- ☐ Import selected stored procedures and functions into the entity model

Model Namespace:

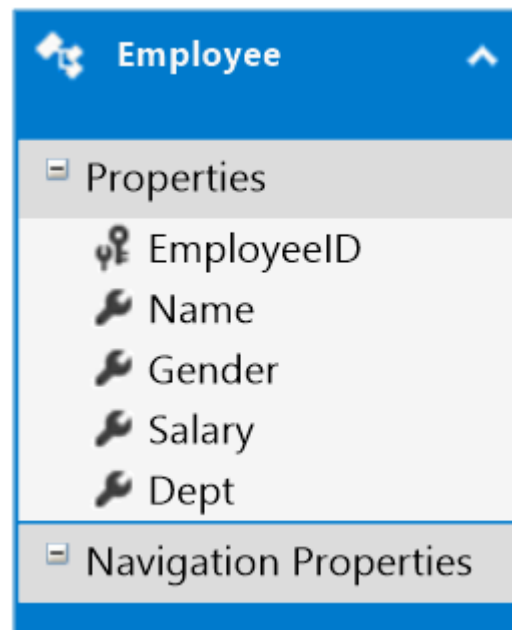
< Previous

Next >

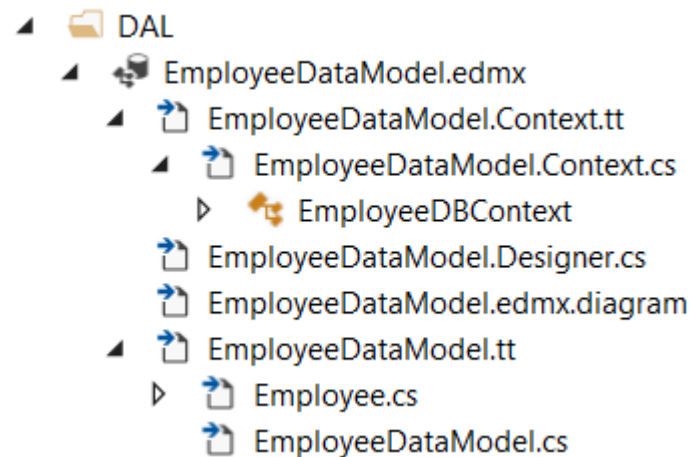
Finish

Cancel

Once you click on the Finish button, then it will create the Employee model as shown below



The Folder structure for the EmployeeDataModel.edmx file as shown below



Following is the auto-generated Employee entity generated by Entity Framework

```
namespace RepositoryUsingEFinMVC.DAL
{
    public partial class Employee
    {
        public int EmployeeID { get; set; }
        public string Name { get; set; }
        public string Gender { get; set; }
        public Nullable<int> Salary { get; set; }
        public string Dept { get; set; }
    }
}
```

Following is auto-generated Context class i.e. EmployeeDBContext generated by Entity Framework

```
namespace RepositoryUsingEFinMVC.DAL
{
    public partial class EmployeeDBContext : DbContext
    {
        public EmployeeDBContext()
            : base("name=EmployeeDBContext")
        {
        }
        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            throw new UnintentionalCodeFirstException();
        }
        public virtual DbSet<Employee> Employees { get; set; }
    }
}
```

Once we create the ADO.NET Entity Data model the next step is to create the Employee Repository for our application.

STEP4: Creating Employee Repository

A repository typically does at least five operations –

1. Selecting all records from a table
2. Selecting a single record based on its primary key
3. Insert
4. Update
5. Delete

This list, however, is not fixed. You may have more or fewer methods in the repository. For the sake of our example let's decide that these five operations are needed from the Employee repository.

To achieve this first we will create an Interface (i.e. IEmployeeRepository) with these five methods and then we will implement this interface in a class (i.e. EmployeeRepository)

First, add a folder with the name Repository to our project.

To do so, right click on the Project => Add => New Folder

Rename the folder name as Repository

Now add an Interface within the Repository folder with the name IEmployeeRepository.cs and copy and paste the below code.

IEmployeeRepository.cs

```
using RepositoryUsingEFinMVC.DAL;
using System.Collections.Generic;
namespace RepositoryUsingEFinMVC.Repository
{
    public interface IEmployeeRepository
    {
        IEnumerable<Employee> GetAll();
    }
}
```

```
Employee GetById(int EmployeeID);  
void Insert(Employee employee);  
void Update(Employee employee);  
void Delete(int EmployeeID);  
void Save();  
}  
}
```

The IEmployeeRepository interface has five methods as listed below:

GetAll(): This [method](#) is used to return all the Employee entities as an enumerable collection (such as a generic List).

GetById(): This method accepts an integer parameter representing an Employee ID (EmployeeID is an integer column in the Employee table in the database) and returns a single Employee entity matching that Employee ID.

Insert(): This method accepts an Employee object as the parameter and adds that Employee object to the Employees DbSet.

Update(): This method accepts an Employee object as parameter and marks that Employee object as a modified Employee in the DbSet.

Delete(): This method accepts an EmployeeID as a parameter and removes that Employee entity from the Employees DbSet.

Save(): This method saves the changes to the EmployeeDB database.

Next, add EmployeeRepository class and implement IEmployeeRepository in it.

To do so, add a class file within the Repository folder with the name EmployeeRepository.cs and copy and paste the below code.

EmployeeRepository.cs

```
using RepositoryUsingEFInMVC.DAL;  
using System;  
using System.Collections.Generic;
```

```
using System.Data.Entity;
using System.Linq;

namespace RepositoryUsingEFInMVC.Repository
{
    public class EmployeeRepository : IEmployeeRepository
    {
        private readonly EmployeeDBContext _context;

        public EmployeeRepository()
        {
            _context = new EmployeeDBContext();
        }

        public EmployeeRepository(EmployeeDBContext context)
        {
            _context = context;
        }

        public IEnumerable<Employee> GetAll()
        {
            return _context.Employees.ToList();
        }

        public Employee GetById(int EmployeeID)
        {
            return _context.Employees.Find(EmployeeID);
        }

        public void Insert(Employee employee)
        {
            _context.Employees.Add(employee);
        }

        public void Update(Employee employee)
        {

```

```
        _context.Entry(employee).State = EntityState.Modified;
    }
    public void Delete(int EmployeeID)
    {
        Employee employee = _context.Employees.Find(EmployeeID);
        _context.Employees.Remove(employee);
    }

    public void Save()
    {
        _context.SaveChanges();
    }
    private bool disposed = false;

    protected virtual void Dispose(bool disposing)
    {
        if (!this.disposed)
        {
            if (disposing)
            {
                _context.Dispose();
            }
        }
        this.disposed = true;
    }

    public void Dispose()
    {
        Dispose(true);

        GC.SuppressFinalize(this);
    }
}
```

The `EmployeeRepository` class implements all the five methods discussed above. Notice that it has two constructor definitions – one that takes no parameters and the one that takes the data context instance as the parameter. The second version will be useful when you wish to pass the context from outside (such as during testing or while using the Unit of Work pattern). We will discuss this in details when we discuss Unit Of Work concepts in a later article.

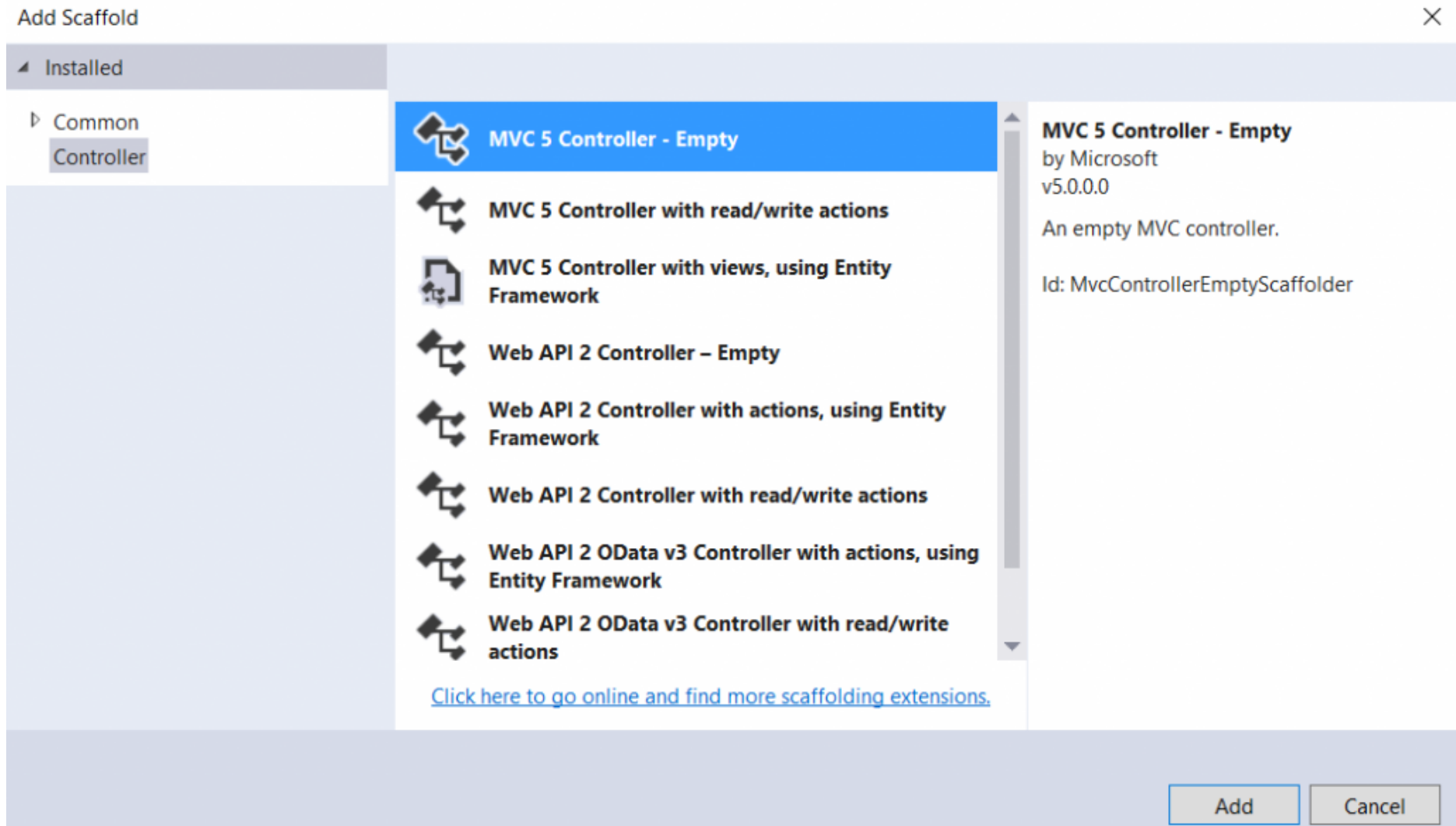
STEP5: Using Employee Repository in a Controller

We already created our Employee Repository. Let's use this Employee Repository in a controller.

To do so, add a controller class inside the Controllers folder and name it `EmployeeController`.

Right click on Controllers Folder and select Add => Controller

Then select MVC5 Controller – Empty as shown below.



Once we click on Add button one popup will open for providing the Controller name as shown below.

Add Controller ✕

Controller name:

Provide the controller name as EmployeeController and click on ADD button which will add the Employee Controller within the Controllers folder.

Now, copy and paste the below code in Employee Controller.

```
using System.Web.Mvc;
using RepositoryUsingEFinMVC.DAL;
using RepositoryUsingEFinMVC.Repository;
namespace RepositoryUsingEFinMVC.Controllers
{
    public class EmployeeController : Controller
    {
        private IEmployeeRepository _employeeRepository;

        public EmployeeController()
        {
            _employeeRepository = new EmployeeRepository(new EmployeeDbContext());
        }
        public EmployeeController(IEmployeeRepository employeeRepository)
        {
            _employeeRepository = employeeRepository;
        }

        [HttpGet]
        public ActionResult Index()
        {
            var model = _employeeRepository.GetAll();
        }
    }
}
```

```
        return View(model);
    }

    [HttpGet]
    public ActionResult AddEmployee()
    {
        return View();
    }

    [HttpPost]
    public ActionResult AddEmployee(Employee model)
    {
        if (ModelState.IsValid)
        {
            _employeeRepository.Insert(model);
            _employeeRepository.Save();
            return RedirectToAction("Index", "Employee");
        }
        return View();
    }

    [HttpGet]
    public ActionResult EditEmployee(int EmployeeId)
    {
        Employee model = _employeeRepository.GetById(EmployeeId);
        return View(model);
    }

    [HttpPost]
    public ActionResult EditEmployee(Employee model)
    {
        if (ModelState.IsValid)
        {
            _employeeRepository.Update(model);
            _employeeRepository.Save();
        }
    }
}
```

```
        return RedirectToAction("Index", "Employee");
    }
    else
    {
        return View(model);
    }
}

[HttpGet]
public ActionResult DeleteEmployee(int EmployeeId)
{
    Employee model = _employeeRepository.GetById(EmployeeId);
    return View(model);
}

[HttpPost]
public ActionResult Delete(int EmployeeID)
{
    _employeeRepository.Delete(EmployeeID);
    _employeeRepository.Save();
    return RedirectToAction("Index", "Employee");
}
}
}
```

The Employee controller has two versions of the constructor and seven action methods. Notice that there is a private variable of type `IEmployeeRepository` at the class level. The parameterless constructor sets this variable to an instance of `EmployeeRepository`. The other version of the constructor accepts an implementation of `IEmployeeRepository` from the external world and sets it to the private variable. This second version is useful during testing where you will supply a mock implementation of Employee repository from the test project.

The seven methods defined by the Employee controller are as follows:

Index(): This action method displays Index view and passes a List of Employee entities as its model.

AddEmployee(): Displays the Add employee view.

AddEmployee(Employee model): Add Employee view submits data to this method. It receives the data as an Employee instance and then inserts an Employee using the repository.

EditEmployee(int EmployeeId): Displays the Edit Employee view. It accepts an Employee ID as the parameter and populates the Edit Employee view with the data of the existing Employee whose ID it accepts as the parameter.

EditEmployee(Employee model): Edit Employee view submits the data to this method. It receives the data as an Employee instance and then updates the Employee using the repository.

DeleteEmployee(int EmployeeId): Displays the Delete Employee view.

Delete(int EmployeeId): Delete Employee view submits the data to this action method. The action then deletes the Employee using the repository.

STEP6: Adding Views:

1. Index.cshtml View

```
@model IEnumerable<RepositoryUsingEFinMVC.DAL.Employee>
@{
    ViewBag.Title = "Index";
}
<h2>Index</h2>
<p>
    @Html.ActionLink("Add Employee", "AddEmployee")
</p>
<table class="table">
    <tr>
        <th>
            @Html.DisplayNameFor(model => model.Name)
        </th>
```

```
<th>
    @Html.DisplayNameFor(model => model.Gender)
</th>
<th>
    @Html.DisplayNameFor(model => model.Salary)
</th>
<th>
    @Html.DisplayNameFor(model => model.Dept)
</th>
<th></th>
</tr>

@foreach (var item in Model)
{
    <tr>
        <td>
            @Html.DisplayFor(modelItem => item.Name)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Gender)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Salary)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Dept)
        </td>
        <td>
            @Html.ActionLink("Edit", "EditEmployee", new { EmployeeId = item.EmployeeID }) |
            @Html.ActionLink("Delete", "DeleteEmployee", new { EmployeeId = item.EmployeeID })
        </td>
    </tr>
}
</table>
```

2. AddEmployee.cshtml View

```
@model RepositoryUsingEFinMVC.DAL.Employee

@{
    ViewBag.Title = "AddEmployee";
}
@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()

    <div class="form-horizontal">
        <h4>Add Employee</h4>
        <hr />
        @Html.ValidationSummary(true)

        <div class="form-group">
            @Html.LabelFor(model => model.Name, new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Name)
                @Html.ValidationMessageFor(model => model.Name)
            </div>
        </div>

        <div class="form-group">
            @Html.LabelFor(model => model.Gender, new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Gender)
                @Html.ValidationMessageFor(model => model.Gender)
            </div>
        </div>

        <div class="form-group">
            @Html.LabelFor(model => model.Salary, new { @class = "control-label col-md-2" })
```

```

        <div class="col-md-10">
            @Html.EditorFor(model => model.Salary)
            @Html.ValidationMessageFor(model => model.Salary)
        </div>
    </div>

    <div class="form-group">
        @Html.LabelFor(model => model.Dept, new { @class = "control-label col-md-2" })
        <div class="col-md-10">
            @Html.EditorFor(model => model.Dept)
            @Html.ValidationMessageFor(model => model.Dept)
        </div>
    </div>

    <div class="form-group">
        <div class="col-md-offset-2 col-md-10">
            <input type="submit" value="Save" class="btn btn-default" />
        </div>
    </div>
</div>
}

<div>
    @Html.ActionLink("Back to Employee List", "Index")
</div>

@section Scripts {
    @Scripts.Render("~/bundles/jqueryval")
}

```

3. EditEmployee.cshtml View

```

@model RepositoryUsingEFinMVC.DAL.Employee
@{
    ViewBag.Title = "EditEmployee";
}

```

```
}
@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()

    <div class="form-horizontal">
        <h4>Edit Employee</h4>
        <hr />
        @Html.ValidationSummary(true)
        @Html.HiddenFor(model => model.EmployeeID)

        <div class="form-group">
            @Html.LabelFor(model => model.Name, new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Name)
                @Html.ValidationMessageFor(model => model.Name)
            </div>
        </div>

        <div class="form-group">
            @Html.LabelFor(model => model.Gender, new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Gender)
                @Html.ValidationMessageFor(model => model.Gender)
            </div>
        </div>

        <div class="form-group">
            @Html.LabelFor(model => model.Salary, new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Salary)
                @Html.ValidationMessageFor(model => model.Salary)
            </div>
        </div>
    </div>
}
```



```

<div class="form-group">
    @Html.LabelFor(model => model.Dept, new { @class = "control-label col-md-2" })
    <div class="col-md-10">
        @Html.EditorFor(model => model.Dept)
        @Html.ValidationMessageFor(model => model.Dept)
    </div>
</div>

<div class="form-group">
    <div class="col-md-offset-2 col-md-10">
        <input type="submit" value="Update" class="btn btn-default" />
    </div>
</div>
</div>
}

<div>
    @Html.ActionLink("Back to Employee List", "Index")
</div>

@section Scripts {
    @Scripts.Render("~/bundles/jqueryval")
}

```

4. DeleteEmployee.cshtml View

```

@model RepositoryUsingEFInMVC.DAL.Employee
@{
    ViewBag.Title = "DeleteEmployee";
}
<h3>Are you sure you want to delete this?</h3>
<div>
    @using (Html.BeginForm("Delete", "Employee", FormMethod.Post))
    {

```

```
@Html.HiddenFor(e => e.EmployeeID)
<h4>Delete Employee</h4>
<hr />
<dl class="dl-horizontal">
    <dt>
        @Html.DisplayNameFor(model => model.Name)
    </dt>

    <dd>
        @Html.DisplayFor(model => model.Name)
    </dd>

    <dt>
        @Html.DisplayNameFor(model => model.Gender)
    </dt>

    <dd>
        @Html.DisplayFor(model => model.Gender)
    </dd>

    <dt>
        @Html.DisplayNameFor(model => model.Salary)
    </dt>

    <dd>
        @Html.DisplayFor(model => model.Salary)
    </dd>

    <dt>
        @Html.DisplayNameFor(model => model.Dept)
    </dt>

    <dd>
        @Html.DisplayFor(model => model.Dept)
    </dd>
```

```

</dl>
<div class="form-actions no-color">
    <input type="submit" value="Delete" class="btn btn-default" /> |
    @Html.ActionLink("Back to Employee List", "Index")
</div>
}
</div>

```

Once we created the four views now let's change the default route to Employee Controller and Index action method in the RouteConfig class as shown below.

```

namespace RepositoryUsingEFinMVC
{
    public class RouteConfig
    {
        public static void RegisterRoutes(RouteCollection routes)
        {
            routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

            routes.MapRoute(
                name: "Default",
                url: "{controller}/{action}/{id}",
                defaults: new { controller = "Employee", action = "Index", id =
UrlParameter.Optional }
            );
        }
    }
}

```

Now run the application and perform the CRUD operation and see everything is working as expected.

In the next article, I will discuss Generic Repository Pattern in ASP.NET MVC application using Entity Framework.

SUMMARY

In this article, I try to explain **Basic Repository pattern in ASP.NET** MVC application using Entity Framework step by step with a simple example. I hope this article will help you with your need. I would like to have your feedback. Please post your feedback, question, or comments about this article.

[Factory](#)[Diagram](#)[Free Templates](#)[Mp4 Download](#)[Examples](#)[Image](#)[Models](#)[infolinks](#)**1****5****2****6****3****7****4****8**

5 thoughts on "How to implement Repository Design Pattern in C#"

**LEARNING**

OCTOBER 22, 2018 AT 7:15 PM

Hi,

Good article, but like a lot of these articles it always uses just one table.

Why not make it more "real world" by using an ENUM for Male and Female and another table for the Department name which when then becomes a dropdown list in the create and edit pages?

That would give us a much more complete view of how to build something for real world application.

Thanks

[Reply](#)



ZAHID

MAY 12, 2019 AT 12:40 AM

In a first glance its look fine. Looking foreword for Unit Of Work

[Reply](#)



VOLODYMYR

MAY 13, 2019 AT 2:05 PM

Hi!

How (where) correct use the Dispose method of our EmployeeRepository class?

[Reply](#)



VOLODYMYR

MAY 13, 2019 AT 2:07 PM

* in asp.net mvc application

[Reply](#)

**JOHN**

JUNE 12, 2019 AT 9:31 PM

Does it allow you to use a transaction in case you have multiple repositories and you submit data to database?

[Reply](#)

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name*

Email*

Website

Post Comment

SINGLETON DESIGN PATTERN

- ✓ [Singleton Design Pattern in C#](#)
 - ✓ [Why Singleton Class sealed in C#](#)
 - ✓ [Thread-safe Singleton Design Pattern in C#](#)
 - ✓ [Lazy Loading and Eager loading in Singleton Design Pattern](#)
 - ✓ [Singleton VS Static class in C#](#)
 - ✓ [Singleton Design Pattern Real Time Example in C#](#)
-

DEPENDENCY INJECTION DESIGN PATTERN

- ✓ [Dependency Injection in C#](#)
 - ✓ [Property and Method Dependency Injection in C#](#)
 - ✓ [Dependency Injection using Unity Container in MVC](#)
-

REPOSITORY DESIGN PATTERN

- ✓ [Repository Design Pattern in C#](#)
 - ✓ [How to implement Repository Design Pattern in C#](#)
 - ✓ [Generic Repository Pattern in C#](#)
 - ✓ [Repository Pattern Implementation Guidelines in c#](#)
 - ✓ [Unit of Work using Repository Design Pattern](#)
-

Factory Design Pattern

- ✓ [Factory Design Pattern in C#](#)
 - ✓ [Factory Method Design Pattern in C#](#)
 - ✓ [Abstract Factory Design Pattern in C#](#)
-

INVERSION OF CONTROL

- ✓ [Introduction to Inversion of Control](#)
- ✓ [Inversion of Control Using Factory Pattern in C#](#)
- ✓ [Inversion of Control Using Dependency Inversion Principle](#)

- ✓ [Inversion of Control Using Dependency Injection Design pattern](#)
- ✓ [Inversion of Control Containers in C#](#)

1 | SIMPLE DRESS PATTERNS ▶

2 | FREE JELLY ROLL QUILT PATTERN ▶

3 | WOMEN'S DRESS PATTERNS ▶

4 | EASY HOBO BAG PATTERNS ▶

5 | EMPLOYEE SATISFACTION SURVEY ▶

[Newsletter](#) [Forums](#) [Blog](#) [About](#) [Privacy Policy](#) [Contact](#)

Copyright © 2019 Dot Net Tutorials | Design by Sunrise Pixel