|  |
| --- |
|  |
| LINQ – DbContext |
| Code First Entity Framework |
|  |
|  |
|  |

|  |
| --- |
|  |

# 

LINQ- DbContext

Objectives

After completing this lab, you should understand how to do the following:

* Use a code-first approach to the entity framework
* With with the DbContext API

Overview

In this lab we’ll build an ASP.NET MVC 3.0 application. If ASP.NET MVC 3.0 is not installed in Visual Studio 2010, click the “Install MVC 3” button here: <http://www.asp.net/mvc/mvc3>.

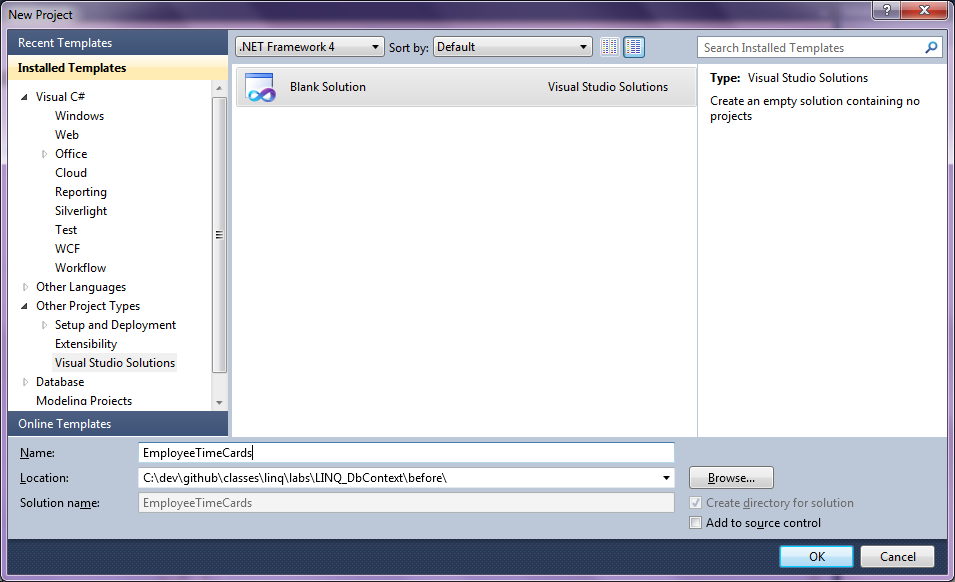
We’ll also be using SQL Server Compact, install it from <http://www.microsoft.com/en-us/download/details.aspx?id=17876>.

You’ll also want to install the SQL Server Compact tools for Visual Studio SP1 here - <http://go.microsoft.com/fwlink/?LinkId=212219>.

# Part I – Create the Project

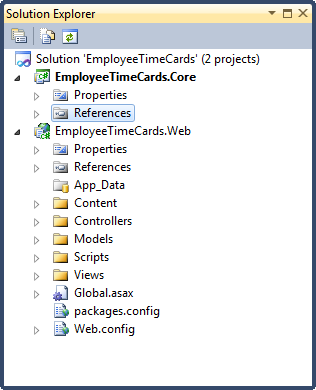
First, we’ll need to setup the project solution. We want to have a class library for our “core” domain logic, and a project for the web application.

1. In Visual Studio, create a new **empty** solution in the before directory of this lab. Name the solution EmployeeTimeCards.



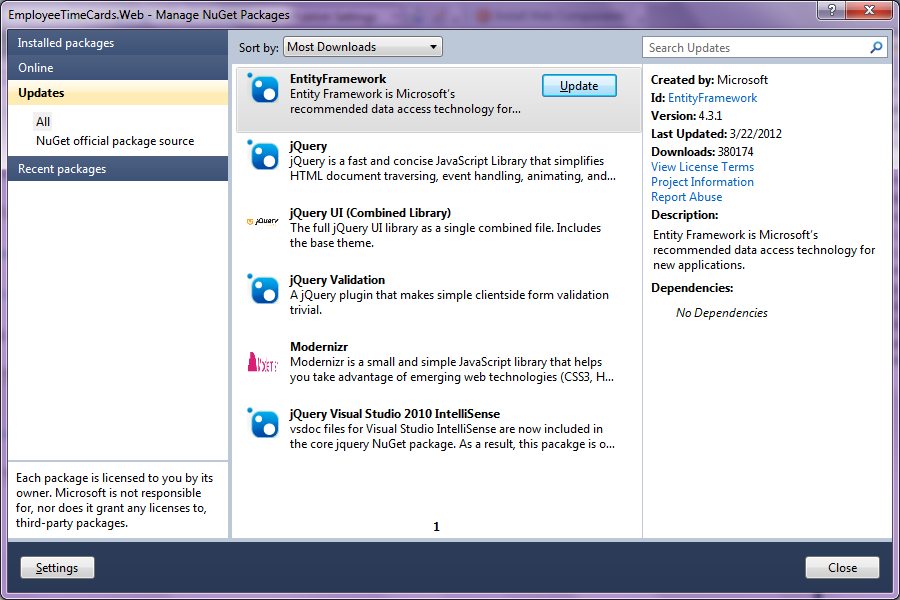
1. Add a new class library project to the solution named “EmployeeTimeCards.Core”.
2. Delete Class1.cs from the new project.
3. Add a new ASP.NET MVC 3 project to the solution named EmployeeTimeCards.Web. Make the application an “Internet Application” using the Razor View engine.

Your solution should look like the following.

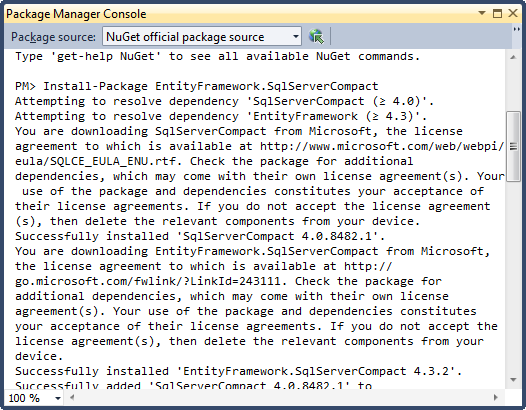


1. Add a reference from the .Web project to the .Core project. (Right-click Refernces in the .Web project, select Add Reference, go to Projects tab, select the .Core project).
2. Update your Nuget package for the Entity Framework (this step is **important** – make sure you are using at least version 4.3 of the Entity Framework).

*To update your package, right click the References node in the .Web project and select “Manage NuGet Packages”. Go to the Updates tab on the left, and you should see the Entity Framework listed with an Update button beside it. Click the button!*



1. Use the Package Manager Console (View -> Other Windows -> Package Manager Console) to install EntityFramework.SqlServerCompact (yes, you can do this through the GUI dialog above, too, but the console window is good to know about). Make sure the **Default Project is EmployeeTimeCards.Web**. Use the command **Install-Package EntityFramework.SqlServerCompact**.



*SQL Server Compact is an embedded database. The database engine will be configured to create database files in the App\_Data folder. Note: this will also update the EntityFramework version if we previously had not updated the version manually.*

1. Build the application to make sure there are no compiler errors.
2. Right-click the .Web project and set it as the Startup Project.
3. Run the application to ensure the project runs and displays a UI in the browser.

*At this point we are setup to write some code.*

## Part II – A Simple Model

*Now we’ll create the domain objects the application will center around.*

1. Add a class named Employee to the .Core project.
2. Give the Employee properties to store an integer Id, first name, last name, and an ICollection of TimeCard (not yet created). Make the properties virtual.

public class Employee

{

public virtual int Id { get; set; }

public virtual string FirstName { get; set; }

public virtual string LastName { get; set; }

public virtual ICollection<TimeCard> TimeCards { get; set; }

}

1. Add a reference from .Core to System.ComponentModel.DataAnnotations. Use the [Required] attribute on the FirstName and LastName properties of an Employee.

public class Employee

{

public virtual int Id { get; set; }

**[Required]**

public virtual string FirstName { get; set; }

**[Required]**

public virtual string LastName { get; set; }

public virtual ICollection<TimeCard> TimeCards { get; set; }

}

1. Create a TimeCard class to in Core to store an Id (int), HoursWorked (int), Employee, and PayPeriod (not yet created).

public class TimeCard

{

public virtual int Id { get; set; }

public virtual int Hours { get; set; }

public virtual PayPeriod Period { get; set; }

public virtual Employee Employee { get; set; }

}

1. Create a PayPeriod class to store Id, StartDate, and EndDate (dates are DateTime types).

public class PayPeriod

{

public virtual int Id { get; set; }

public virtual DateTime StartDate { get; set; }

}

1. Build the application to make sure there are no compiler errors.

*Next, we’ll create a DbContext derived class to access the database. Note: if you run the project before we complete all of the following steps, you might create the database without all the settings required for it to work. If this happens, or for some other reason you get an exception about the “model changed” – stop the web server, go to the App\_Data folder, and delete the database files.*

1. **In the .Web project**, create a folder named Infrastructure.
2. Add a class in the infrastructure project named TimeCardContext. Make the class derive from DbContext.
3. Give the TimeCardContext public properties of type DbSet<T> for Employee and PayPeriod (where T will be the concrete type for employees and pay periods).
4. In the context class, override OnModelCreating to set the maximum length of the FirstName value for an employee to 80 characters.

public class TimeCardContext : DbContext

{

public DbSet<Employee> Employees { get; set; }

public DbSet<PayPeriod> PayPeriods { get; set; }

protected override void OnModelCreating(

DbModelBuilder modelBuilder)

{

modelBuilder.Entity<Employee>()

.Property(e => e.FirstName)

.HasMaxLength(80);

base.OnModelCreating(modelBuilder);

}

}

*Next, we are going to add an “Initializer” to populate the database with some lookup data we’ll need immediately.*

1. Add a class named TimeCardContextInitializer in the new folder.
2. Make the new class implement IDatabaseInitializer<TimeCardContext>. (IDatabaseInitializer is in the System.Data.Entity namespace).

public class TimeCardContextInitializer :

IDatabaseInitializer<TimeCardContext>

{

public void InitializeDatabase(TimeCardContext context)

{

}

}

1. In the InitializeDatabase method, create some code that will add one PayPeriod object to the database for each month from January 1, 2012 to December 1, 2015.

*These are monthly pay periods. For example, we should see a PayPeriod with a StartDate of Jan 1, 2012, and Feb 1, 2012, and March 1, 2012 – all the way to Dec 1, 2012.*

public class TimeCardContextInitializer :

IDatabaseInitializer<TimeCardContext>

{

public void InitializeDatabase(TimeCardContext context)

{

context.Database.CreateIfNotExists();

var date = new DateTime(2012, 1, 1);

var endDate = new DateTime(2015, 12, 31);

while(date < endDate)

{

context.PayPeriods.Add(new PayPeriod {StartDate = date});

date = date.AddMonths(1);

}

context.SaveChanges();

}

}

1. Open the global.asax.cs file.
2. In the Application\_Start method, use Database.SetInitializer to register your TimeCardContextInitializer.

protected void Application\_Start()

{

AreaRegistration.RegisterAllAreas();

**Database.SetInitializer(new TimeCardContextInitializer());**

RegisterGlobalFilters(GlobalFilters.Filters);

RegisterRoutes(RouteTable.Routes);

}

1. Build the project to make sure everything is working correctly.

# Part III – Adding UI to Create And List Employees

1. In the .Web project, go to the Views/Home folder and delete the Index.cshtml view.
2. Open the Controllers/HomeController.cs file and delete the existing code in the Index action.
3. Place code in the Index action to retrieve all Employees in the database and pass them as the model to the default view.

public ActionResult Index()

{

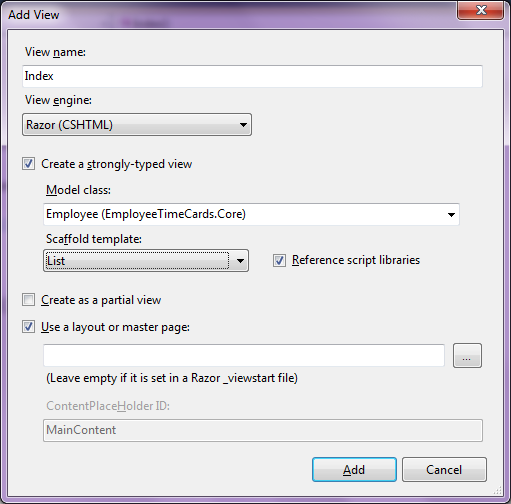
var db = new TimeCardContext();

var model = db.Employees;

return View(model);

}

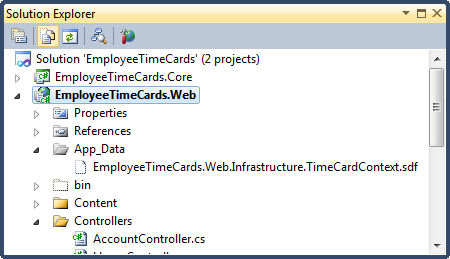
1. Right-click inside the Index action and select Add View. Create a strongly typed view for Employee, and use the List Scaffold template.



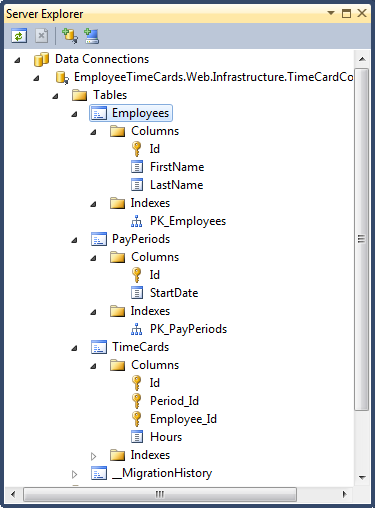
1. **Run** the application.

*The view should not display any records. Although we don’t have any employees in the database, this should create a database in the App\_Data folder at runtime.*

1. Click the .Web project in Solution Explorer, and then click the Show All Files button in the Solution Explorer toolbar (see top arrow below).
2. Open the App\_Data folder and double click the .sdf file inside.



1. The .sdf should open in the Server Explorer window. Verify you have four tables in the .sdf (one for each of our model objects, plus a migration history table).



1. Right-click the PayPeriods table and click Show Data. Note how your initializer placed PayPeriod data into the table.
2. Right-click the Employees table and select Edit Schema. Note how the FirstName column has a length of 80 characters.
3. Right-click the TimeCards table and select Table Properties. Go to Manage Relations. Note how there are foreign key relationships established to the Employees and PayPeriods table.
4. Return to Controllers\HomeController.cs.
5. Add a Create method that returns a default View. Give it an [HttpGet] attribute.
6. Add a second Create method that takes an Employee as a parameter and has an [HttpPost] parameter. Inside the method, save the employee in the database if ModelState.IsValid is true, and redirect back to the Index action. Otherwise, return the same default view.

[HttpGet]

public ActionResult Create()

{

return View();

}

[HttpPost]

public ActionResult Create(Employee newEmployee)

{

if(ModelState.IsValid)

{

var db = new TimeCardContext();

db.Employees.Add(newEmployee);

db.SaveChanges();

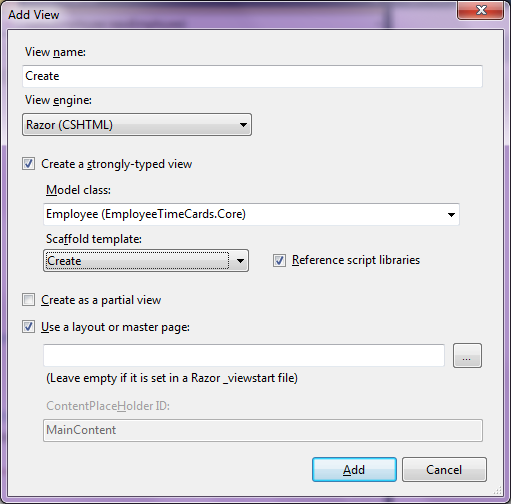
return RedirectToAction("Index");

}

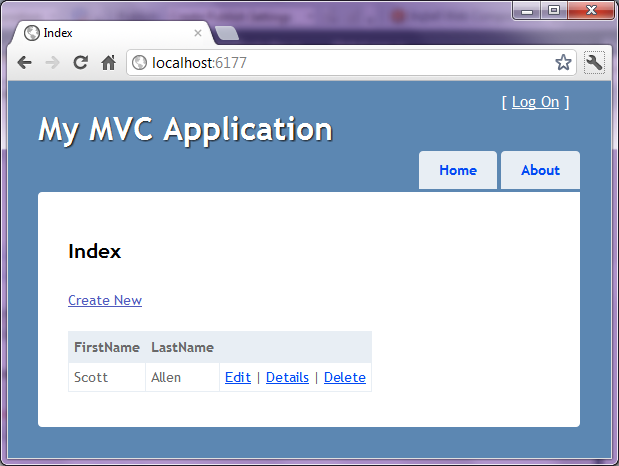
return View(newEmployee);

}

1. Right-click in one of the two Create actions and select Add View. Create a strongly typed view for Employee with the Create scaffold action.



1. Run the application and see if you can create an employee. Make sure the employee starts to appear in the Index view.



### Big Warning About Over Posting Attacks

There is a possible security vulnerability with the code we just wrote. Although it won’t happen in all scenarios, you should be aware of mass assignment attacks when using ASP.NET MVC model binding directly against entities. We’re doing it here to keep things easy.

Read <http://odetocode.com/blogs/scott/archive/2012/03/11/complete-guide-to-mass-assignment-in-asp-net-mvc.aspx> for more details, or ask the instructor about potential problems with the code we just wrote.

# Part IV – Edit

1. In HomeController.cs, add an Edit action with an HttpGet attribute that will display a single employee in an Edit form. The action will take an id parameter to look up the employee, and you can use a LINQ operator (.First, for example), or the .Find method to get the employee.

[HttpGet]

public ActionResult Edit(int id)

{

var db = new TimeCardContext();

var model = db.Employees.Find(id);

return View(model);

}

1. Implement an Edit action for the HttpPost that will save an employee in the database. Note: you’ll want to “attach” the object to the DbSet, not add it to the DbSet.

[HttpPost]

public ActionResult Edit(Employee updatedEmployee)

{

if(ModelState.IsValid)

{

var db = new TimeCardContext();

db.Entry(updatedEmployee).State = EntityState.Modified;

db.SaveChanges();

return RedirectToAction("Index");

}

return View(updatedEmployee);

}

1. Scaffold an Edit view that is strongly typed to Employee.
2. Run the application to verify you can edit an existing Employee.

# Conclusion

Hooray! You’ve now done a bit of work with the Entity Framework DbContext API. If you are up for a challenge, implement a “Detail” and “Delete” view for Employee.