

Long Le's Blog

Measuring programming progress by lines of code is like measuring aircraft building progress by weight. – B. Gates

[RSS Feed](#) [Twitter](#)

May 11, 2013

Generically Implementing the Unit of Work & Repository Pattern with Entity Framework in MVC & Simplifying Entity Graphs

112 Comments

This will be part two of a six part series of blog posts.

1. [Modern Web Application Layered High Level Architecture with SPA, MVC, Web API, EF, Kendo UI, OData](http://blog.longle.net/2013/06/13/modern-web-application-layered-high-level-architecture-with-spa-mvc-web-api-ef/) (<http://blog.longle.net/2013/06/13/modern-web-application-layered-high-level-architecture-with-spa-mvc-web-api-ef/>)
2. [Generically Implementing the Unit of Work & Repository Pattern with Entity Framework in MVC & Simplifying Entity Graphs](http://blog.longle.net/2013/05/11/genericizing-the-unit-of-work-pattern-repository-pattern-with-entity-framework-in-mvc/) (<http://blog.longle.net/2013/05/11/genericizing-the-unit-of-work-pattern-repository-pattern-with-entity-framework-in-mvc/>)
3. [MVC 4, Kendo UI, SPA with Layout, View, Router & MVVM](http://blog.longle.net/2013/06/17/mvc-4-kendo-ui-spa-with-layout-router-mvvm/) (<http://blog.longle.net/2013/06/17/mvc-4-kendo-ui-spa-with-layout-router-mvvm/>)
4. [MVC 4, Web API, OData, EF, Kendo UI, Grid, Datasource \(CRUD\) with MVVM](http://blog.longle.net/2013/06/18/mvc-4-web-api-odata-entity-framework-kendo-ui-grid-datasource-with-mvvm/) (<http://blog.longle.net/2013/06/18/mvc-4-web-api-odata-entity-framework-kendo-ui-grid-datasource-with-mvvm/>)
5. [MVC 4, Web API, OData, EF, Kendo UI, Binding a Form to Datasource \(CRUD\) with MVVM](http://blog.longle.net/2013/06/19/mvc-4-web-api-odata-ef-kendo-ui-binding-a-form-to-datasource-crud-with-mvvm-part/) (<http://blog.longle.net/2013/06/19/mvc-4-web-api-odata-ef-kendo-ui-binding-a-form-to-datasource-crud-with-mvvm-part/>)
6. [Upgrading to Async with Entity Framework, MVC, OData AsyncEntitySetController, Kendo UI, Glimpse & Generic Unit of Work Repository Framework v2.0](http://blog.longle.net/2013/10/09/upgrading-to-async-with-entity-framework-mvc-odata-asyncentitysetcontroller-kendo-ui-glimpse-generic-unit-of-work-repository-framework-v2-0/) (<http://blog.longle.net/2013/10/09/upgrading-to-async-with-entity-framework-mvc-odata-asyncentitysetcontroller-kendo-ui-glimpse-generic-unit-of-work-repository-framework-v2-0/>)

Update: 09/18/2013 – Sample application and source code has been uploaded to CodePlex:

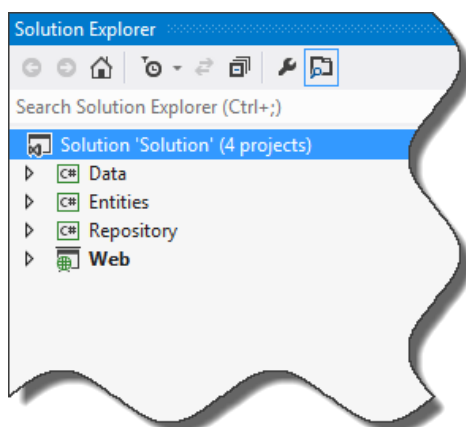
<https://genericunitofworkandrepositories.codeplex.com> (<https://genericunitofworkandrepositories.codeplex.com>), updated project to VS2013, Twitter Bootstrap, MVC 5, EF6, Kendo UI Bootstrap theme, project redeployed to Windows Azure Website.

Update: 07/30/2013 – To see this implementation with DI & IoC with EntLib Unity v3.0, see post: [Bounded DbContext with Generic Unit of Work, Generic Repositories, Entity Framework 6 & EntLib Unity 3.0 in MVC 4](http://blog.longle.net/2013/07/30/bounded-dbcontext-with-generic-unit-of-work-generic-repositories-entity-framework-6-unity-3-0-in-mvc-4/) (<http://blog.longle.net/2013/07/30/bounded-dbcontext-with-generic-unit-of-work-generic-repositories-entity-framework-6-unity-3-0-in-mvc-4/>).

Update: 06/21/2013 – Bug fix: Issue with deleting objects by Id in Repository.Delete(object Id). Updated blog post, and sample solution and added live demo link.

Live demo: <http://longle.azurewebsites.net/Spa/Product#/list> (<http://longle.azurewebsites.net/Spa/Product#/list>)

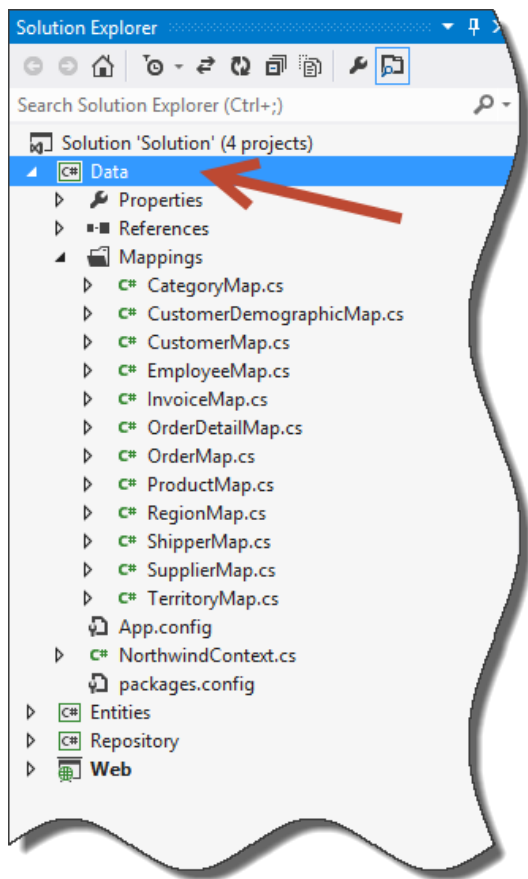
First off let's elegantly setup our solution, and prep it for real world development. We have our solution broken up into four different projects, now let's talk about the "why?".



(<http://lelong37.files.wordpress.com/2013/05/5-8-2013-9-08-17-pm.png>)

Data Project (Data Access Layer)

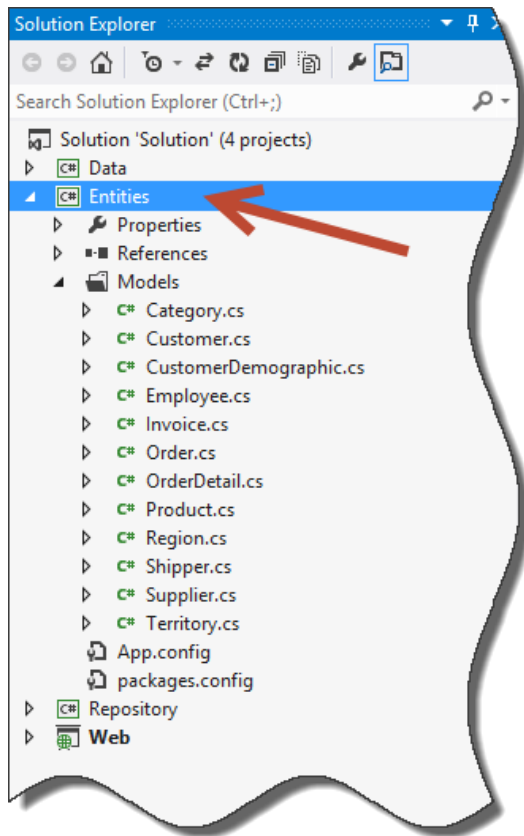
This is where all of our ORM tooling related objects reside. In our case the EF (Entity Framework 6.0 Alpha 3) DataContext, Mappings, Migrations, etc. This gives nice separation, control and isolation of where any persistence related objects live. If ever, one day we need to change the tool of choice, or even upgrade, there's only one layer or project to do this in, the Data project.



<http://lelong37.files.wordpress.com/2013/05/5-8-2013-10-08-26-pm.png>

Entities Project (Domain Layer)

The Entities project is where all of our **POCO** (Plain Old C# Objects) objects will live. POCO's should be very ignorant objects that pretty much have nothing in them but the structure of your data. With that being said, typically anything outside our Repository layer e.g. presentation layer (MVC), services layer (will cover in [next post \(http://blog.longle.net/2013/05/17/generically-implementing-the-unit-of-work-repository-pattern-with-entity-framework-in-mvc-simplifying-entity-graphs-part-2-mvc-4-di-ioc-with-mef-attributeless-conventions/\)](http://blog.longle.net/2013/05/17/generically-implementing-the-unit-of-work-repository-pattern-with-entity-framework-in-mvc-simplifying-entity-graphs-part-2-mvc-4-di-ioc-with-mef-attributeless-conventions/)) should be completely ignorant to any persistence tool or technology e.g. NHibernate, eXpressPersistent, OpenAccess, EF (our case), etc.



<http://lelong37.files.wordpress.com/2013/05/5-8-2013-10-09-31-pm.png>

Repository (Layer)

This is where our **UoW** (Unit of Work) pattern will be implemented as well as our Repository implementation. Our UoW implementation will handle most of our usual CRUD activities.

Two important objectives we will try to with our UoW pattern implementation are:

1. Abstract away the ORM tool, in our case EF.
2. Ensuring that all interactions with the database are happening under one DbContext instance per page request.

Obviously there are many other benefits, such giving us the ability to implement different variations of our UoW, potentially wire up to different types of repositories. For purposes of this article, we'll stake focus on our two primary objectives, and I'll cover the other benefits in later posts.

Web Project (Presentation Layer)

This is our presentation layer, for the purposes of the blog, we will use **MVC** (ASP.NET MVC 4). Again, this project should not have any dependent code on EF assembly, therefore that should not be any references to the EF assembly, it should only reference our Repository project for data access.

Refactoring the NorthwindContext for an Abstracted and Cleaner Implementation

Now that we've gone over the solution and it's projects, let's do a little bit of refactoring and cleaning up with our EF code.

Data.NorthwindDataContext.cs

Before:

```

1  public class NorthwindContext : DbContext
2  {
3      static NorthwindContext()
4      {
5          Database.SetInitializer<NorthwindContext>(null);
6      }
7
8      public NorthwindContext()
9          : base("Name=NorthwindContext")
10     {
11     }
12
13     public DbSet Category Categories { get; set; }
14     public DbSet CustomerDemographic CustomerDemographics { get; set; }
15     public DbSet Customer Customers { get; set; }
16     public DbSet Employee Employees { get; set; }

```

```

17 public DbSet OrderDetail Order_Details { get; set; }
18 public DbSet Order Orders { get; set; }
19 public DbSet Product Products { get; set; }
20 public DbSet Region Regions { get; set; }
21 public DbSet Shipper Shippers { get; set; }
22 public DbSet Supplier Suppliers { get; set; }
23 public DbSet Territory Territories { get; set; }
24 public DbSet Invoice Invoices { get; set; }
25
26
27 protected override void OnModelCreating(DbModelBuilder modelBuilder)
28 {
29     modelBuilder.Configurations.Add(new CategoryMap());
30     modelBuilder.Configurations.Add(new CustomerDemographicMap());
31     modelBuilder.Configurations.Add(new CustomerMap());
32     modelBuilder.Configurations.Add(new EmployeeMap());
33     modelBuilder.Configurations.Add(new Order_DetailMap());
34     modelBuilder.Configurations.Add(new OrderMap());
35     modelBuilder.Configurations.Add(new ProductMap());
36     modelBuilder.Configurations.Add(new RegionMap());
37     modelBuilder.Configurations.Add(new ShipperMap());
38     modelBuilder.Configurations.Add(new SupplierMap());
39     modelBuilder.Configurations.Add(new TerritoryMap());
40     modelBuilder.Configurations.Add(new InvoiceMap());
41 }
42 }

```

After:

```

1 public class NorthwindContext : DbContext, IDbContext
2 {
3     static NorthwindContext()
4     {
5         Database.SetInitializer<NorthwindContext>(null);
6     }
7
8     public NorthwindContext()
9         : base("Name=NorthwindContext")
10    {
11    }
12
13    public new IDbSet<T> Set<T>() where T : class
14    {
15        return base.Set<T>();
16    }
17
18    public override int SaveChanges()
19    {
20        this.ApplyStateChanges();
21        return base.SaveChanges();
22    }
23
24    protected override void OnModelCreating(DbModelBuilder modelBuilder)
25    {
26        modelBuilder.Configurations.Add(new CategoryMap());
27        modelBuilder.Configurations.Add(new CustomerDemographicMap());
28        modelBuilder.Configurations.Add(new CustomerMap());
29        modelBuilder.Configurations.Add(new EmployeeMap());
30        modelBuilder.Configurations.Add(new Order_DetailMap());
31        modelBuilder.Configurations.Add(new OrderMap());
32        modelBuilder.Configurations.Add(new ProductMap());
33        modelBuilder.Configurations.Add(new RegionMap());
34        modelBuilder.Configurations.Add(new ShipperMap());
35        modelBuilder.Configurations.Add(new SupplierMap());
36        modelBuilder.Configurations.Add(new TerritoryMap());
37        modelBuilder.Configurations.Add(new InvoiceMap());
38    }
39 }

```

We can see that our DbContext is now much cleaner, and that it implements IDbContext. IDbContext will be the abstraction we will be working with when interacting with its concrete implementation, NorthwindContext.

Best Practice, Coding Against Abstractions or Interfaces

Abstractions serve as a nice flexibility point later, allowing us to implement different variations of the abstraction (interface). This will be very useful later when we implement **DI** (Dependency Injection) and **IoC** (Inverse of Control) patterns. Coding to an abstraction will also help us easily create unit test, allowing us to inject faked or mocked instances as well. If you're a bit unclear on how this helps set stage for DI, IoC and Unit Testing, no worries, I'll cover these topics in the [next post](http://blog.longle.net/2013/05/17/generically-implementing-the-unit-of-work-repository-pattern-with-entity-framework-in-mvc-simplifying-entity-graphs-part-2-mvc-4-di-ioc-with-mef-attributeless-conventions/) (<http://blog.longle.net/2013/05/17/generically-implementing-the-unit-of-work-repository-pattern-with-entity-framework-in-mvc-simplifying-entity-graphs-part-2-mvc-4-di-ioc-with-mef-attributeless-conventions/>).

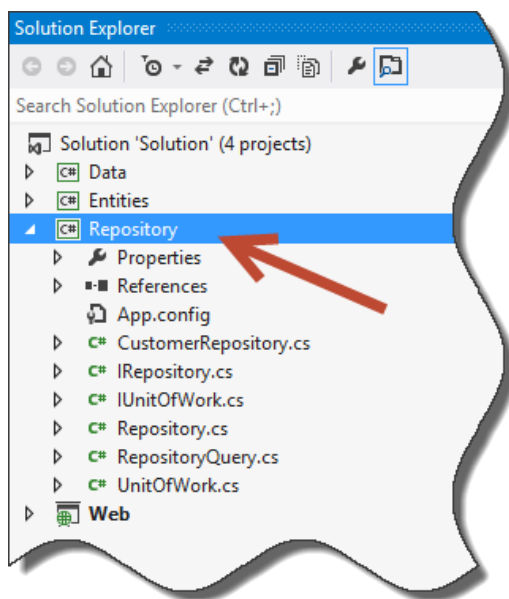
Data.IDbContext.cs

```

1 namespace Data
2 {
3     public interface IDbContext
4     {
5         IDbSet<T> Set<T>() where T : class;
6         int SaveChanges();
7         DbEntityEntry Entry(object o);
8         void Dispose();
9     }
10 }

```

Now, let's take a look at what's all in our Repository project, where our generic extensible repositories will reside.



(<http://lalong37.files.wordpress.com/2013/05/5-10-2013-7-02-10-pm.png>)

IUnitOfWork This is simply the contract or abstraction that we will be working with, when interacting with its concrete implementation which will be UnitOfWork object.

Repository.IUnitOfWork.cs

```

1 namespace Repository
2 {
3     public interface IUnitOfWork
4     {
5         void Dispose();
6         void Save();
7         void Dispose(bool disposing);
8         IRepository<T> Repository<T>() where T : class;
9     }
10 }

```

Concrete Implementation of IUnitOfWork.cs

```

1 namespace Repository
2 {
3     public class UnitOfWork : IUnitOfWork
4     {
5         private readonly IDbContext _context;
6
7         private bool _disposed;
8         private Hashtable _repositories;
9
10        public UnitOfWork(IDbContext context)

```

```

11     {
12         _context = context;
13     }
14
15     public UnitOfWork()
16     {
17         _context = new NorthwindContext();
18     }
19
20     public void Dispose()
21     {
22         Dispose(true);
23         GC.SuppressFinalize(this);
24     }
25
26     public void Save()
27     {
28         _context.SaveChanges();
29     }
30
31     public virtual void Dispose(bool disposing)
32     {
33         if (!_disposed)
34             if (disposing)
35                 _context.Dispose();
36
37         _disposed = true;
38     }
39
40     public IRepository<T> Repository<T>() where T : class
41     {
42         if (_repositories == null)
43             _repositories = new Hashtable();
44
45         var type = typeof(T).Name;
46
47         if (!_repositories.ContainsKey(type))
48         {
49             var repositoryType = typeof(Repository<>);
50
51             var repositoryInstance =
52                 Activator.CreateInstance(repositoryType
53                     .MakeGenericType(typeof(T)), _context);
54
55             _repositories.Add(type, repositoryInstance);
56         }
57
58         return (IRepository<T>) _repositories[type];
59     }
60 }
61 }

```

Let's take a look at our `IRepository Repository()` method here in our `UnitOfWork` implementation. Here we are storing all the activated instances of repositories for each and every requests. One there is a request for a given repository we will first check to see if our `Hashtable` (container to hold all of our activated repository instances) has been created, if not, will go ahead and create our container. Next, we'll scan our container to see if the requested repository instance has already been created, if it has, then will return it, if it hasn't, we will activate the requested repository instance, store it in our container, and then return it. If it helps, you can think of this as lazy loading our repository instances, meaning we are only creating repository instances on demand, this allows us to only create the repository instances needed for a given web request. Last but not least, notice here how we are following best practices mentioned earlier, we are not return the concrete implementation for the `Repository`, but the abstraction, `IRepository`.

Repository.IRepository.cs

```

1 namespace Repository
2 {
3     public interface IRepository<TEntity> where TEntity : class
4     {
5         TEntity FindById(object id);
6         void InsertGraph(TEntity entity);
7         void Update(TEntity entity);
8         void Delete(object id);
9         void Delete(TEntity entity);
10        void Insert(TEntity entity);
11        RepositoryQuery<TEntity> Query();
12    }
13 }

```

Repository.Repository.cs

```

1  public class Repository<TEntity> : IRepository<TEntity> where TEntity : class
2  {
3      internal IDbContext Context;
4      internal IDbSet<TEntity> DbSet;
5
6      public Repository(IDbContext context)
7      {
8          Context = context;
9          DbSet = context.Set<TEntity>();
10     }
11
12     public virtual TEntity FindById(object id)
13     {
14         return DbSet.Find(id);
15     }
16
17     public virtual void InsertGraph(TEntity entity)
18     {
19         DbSet.Add(entity);
20     }
21
22     public virtual void Update(TEntity entity)
23     {
24         DbSet.Attach(entity);
25     }
26
27     public virtual void Delete(object id)
28     {
29         var entity = DbSet.Find(id);
30         var objectState = entity as IObjectState;
31         if (objectState != null)
32             objectState.State = ObjectState.Deleted;
33         Delete(entity);
34     }
35
36     public virtual void Delete(TEntity entity)
37     {
38         DbSet.Attach(entity);
39         DbSet.Remove(entity);
40     }
41
42     public virtual void Insert(TEntity entity)
43     {
44         DbSet.Attach(entity);
45     }
46
47     public virtual RepositoryQuery<TEntity> Query()
48     {
49         var repositoryGetFluentHelper =
50             new RepositoryQuery<TEntity>(this);
51
52         return repositoryGetFluentHelper;
53     }
54
55     internal IQueryable<TEntity> Get(
56         Expression<Func<TEntity, bool>> filter = null,
57         Func<IQueryable<TEntity>>,
58         IOOrderedQueryable<TEntity>> orderBy = null,
59         List<Expression<Func<TEntity, object>>>
60             includeProperties = null,
61         int? page = null,
62         int? pageSize = null)
63     {
64         IQueryable<TEntity> query = DbSet;
65
66         if (includeProperties != null)
67             includeProperties.ForEach(i => { query = query.Include(i); });
68
69         if (filter != null)
70             query = query.Where(filter);
71
72         if (orderBy != null)
73             query = orderBy(query);
74
75         if (page != null && pageSize != null)

```



```

76         query = query
77             .Skip((page.Value - 1)*pageSize.Value)
78             .Take(pageSize.Value);
79
80     return query;
81 }
82 }

```

Our generic implementation for Repository allows us to have have all our basic heavy lifting of a Repository out of the box for any one of our Entities. All we have to do is request for the Repository of interest by passing in the Entity e.g.

```
1 | UnitOfWork.Repository<Customer>
```

will give us the Customer Repository with all our out of the box plumbing available.

Let's take a quick look at our Get method in the Repository implementation.

```

1 | internal IEnumerable<TEntity> Get(
2 |     Expression<Func<TEntity, bool>> filter = null,
3 |     Func<IQueryable<TEntity>,
4 |         IOrderedQueryable<TEntity>> orderBy = null,
5 |     List<Expression<Func<TEntity, object>>>
6 |         includeProperties = null,
7 |     int? page = null,
8 |     int? pageSize = null)
9 | {
10 |     IQueryable<TEntity> query = DbSet;
11
12 |     if (includeProperties != null)
13 |         includeProperties.ForEach(i => query.Include(i));
14
15 |     if (filter != null)
16 |         query = query.Where(filter);
17
18 |     if (orderBy != null)
19 |         query = orderBy(query);
20
21 |     if (page != null && pageSize != null)
22 |         query = query
23 |             .Skip((page.Value - 1)*pageSize.Value)
24 |             .Take(pageSize.Value);
25
26
27 |     return query.ToList();
28 | }

```

The Get method here, handles fetching data. It handles querying the data supporting a filtering, ordering, paging, and eager loading of child types, so that we can make one round trip and eager load the entity graph.

We notice here that the method is marked "internal", this is because we only want the Get method here to be accessible to objects with the same assembly, Repository.dll. We will expose the Get method via the Query method and return the RepositoryQuery object to provide a fluent "ish" api, so that's its a bit more easy and intuitive for our developers when querying with our Repository layer. Note, only methods in our RepositoryQuery will actually invoke the internal Get method, again, which is why we went ahead and marked the Get method internal.

Repository.RepositoryQuery.cs (our fluent api helper class)

```

1 | public sealed class RepositoryQuery<TEntity> where TEntity : class
2 | {
3 |     private readonly List<Expression<Func<TEntity, object>>>
4 |         _includeProperties;
5
6 |     private readonly Repository<TEntity> _repository;
7 |     private Expression<Func<TEntity, bool>> _filter;
8 |     private Func<IQueryable<TEntity>,
9 |         IOrderedQueryable<TEntity>> _orderByQueryable;
10 |     private int? _page;
11 |     private int? _pageSize;
12
13 |     public RepositoryQuery(Repository<TEntity> repository)
14 |     {
15 |         _repository = repository;
16 |         _includeProperties =
17 |             new List<Expression<Func<TEntity, object>>>();
18 |     }
19 | }

```



```

20 public RepositoryQuery<TEntity> Filter(
21     Expression<Func<TEntity, bool>> filter)
22 {
23     _filter = filter;
24     return this;
25 }
26
27 public RepositoryQuery<TEntity> OrderBy(
28     Func<IQueryable<TEntity>, IOrderedQueryable<TEntity>> orderBy)
29 {
30     _orderByQueryable = orderBy;
31     return this;
32 }
33
34 public RepositoryQuery<TEntity> Include(
35     Expression<Func<TEntity, object>> expression)
36 {
37     _includeProperties.Add(expression);
38     return this;
39 }
40
41 public IEnumerable<TEntity> GetPage(
42     int page, int pageSize, out int totalCount)
43 {
44     _page = page;
45     _pageSize = pageSize;
46     totalCount = _repository.Get(_filter).Count();
47
48     return _repository.Get(
49         _filter,
50         _orderByQueryable, _includeProperties, _page, _pageSize);
51 }
52
53 public IEnumerable<TEntity> Get()
54 {
55     return _repository.Get(
56         _filter,
57         _orderByQueryable, _includeProperties, _page, _pageSize);
58 }
59 }

```

Addressing IRepository<TEntity> Extensibility

Well, what happens if we need extra methods a specific Repository? Meaning, how do we address “extensibility” in our Repository? No problem, we have a couple of options here, we can simply inherit a Repository and add your own methods to it, or what I prefer, create extension methods e.g. extending IRepository (with some pseudo code for validating an address with UPS).

Repository.CustomerRepository.cs

```

1  /// <summary>
2  /// Extending the IRepository<Customer>
3  /// </summary>
4  public static class CustomerRepository
5  {
6      public static decimal GetCustomerOrderTotalByYear(
7          this IRepository<Customer> customerRepository,
8          int customerId, int year)
9      {
10         return customerRepository
11             .FindById(customerId)
12             .Orders.SelectMany(o => o.OrderDetails)
13             .Select(o => o.Quantity * o.UnitPrice).Sum();
14     }
15
16     /// <summary>
17     /// TODO:
18     /// This should really live in the Services project (Business Layer),
19     /// however, we'll leave it here for now as an example, and migrate
20     /// this in the next post.
21     /// </summary>
22     public static void AddCustomerWithAddressValidation(
23         this IRepository<Customer> customerRepository, Customer customer)
24     {
25         USPSManager m = new USPSManager("YOUR_USER_ID", true);
26         Address a = new Address();
27         a.Address1 = customer.Address;
28         a.City = customer.City;

```

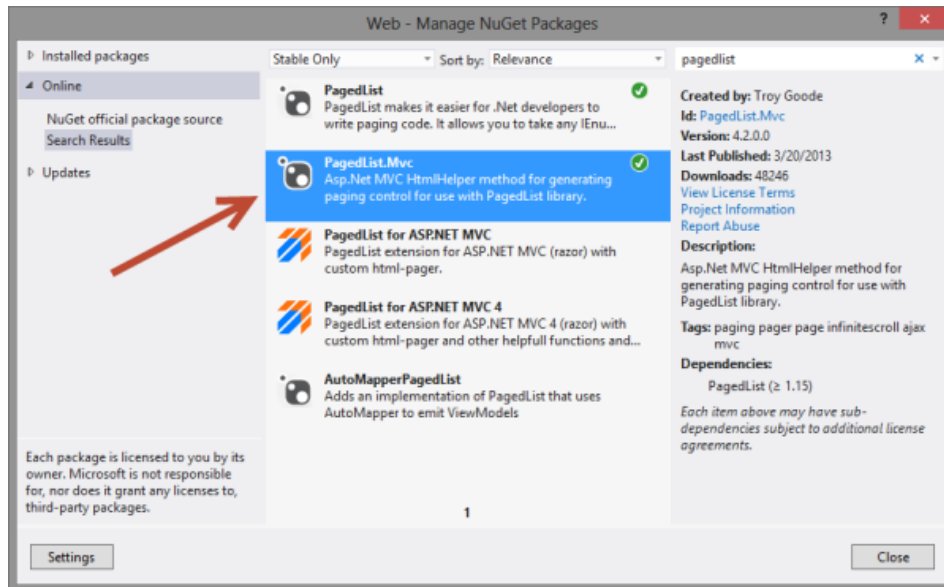
```

29
30     Address validatedAddress = m.ValidateAddress(a);
31
32     if (validatedAddress != null)
33         customerRepository.InsertGraph(customer);
34 }
35 }

```

Great, now that we have our project nicely structured with the our generic implementation of the Unit of Work and Repository Pattern, let's see how we can leverage this by wiring up a simple controller to show a list of customers.

To help us with this go ahead and NuGet the PagedList for MVC so we easily create a view with a paged grid.



(<http://lelong37.files.wordpress.com/2013/05/5-10-2013-6-22-37-pm.png>)

Let's create a **CustomerController Index Action** load a paged list of customers to hydrate a grid.

```

1  public class CustomerController : Controller
2  {
3      public ActionResult Index(int? page)
4      {
5          var pageNumber = page ?? 1;
6          const int pageSize = 20;
7
8          var unitOfWork = new UnitOfWork();
9
10         int totalCustomerCount;
11
12         var customers =
13             unitOfWork.Repository<Customer>()
14                 .Query()
15                 .Include(i => i.CustomerDemographics)
16                 .OrderBy(q => q
17                     .OrderBy(c => c.ContactName)
18                     .ThenBy(c => c.CompanyName))
19                 .Filter(q => q.ContactName != null)
20                 .GetPage(pageNumber, pageSize, out totalCustomerCount);
21
22         ViewBag.Customers = new StaticPagedList<Customer>(
23             customers, pageNumber, pageSize, totalCustomerCount);
24
25         unitOfWork.Save();
26
27         return View();
28     }
29 }

```

Next, let's wire up the **Index.cshtml** view for our CustomerController Index Action.

```

1  @{
2      ViewBag.Title = "Customers";
3  }
4
5  @using PagedList.Mvc;
6  @using PagedList;
7

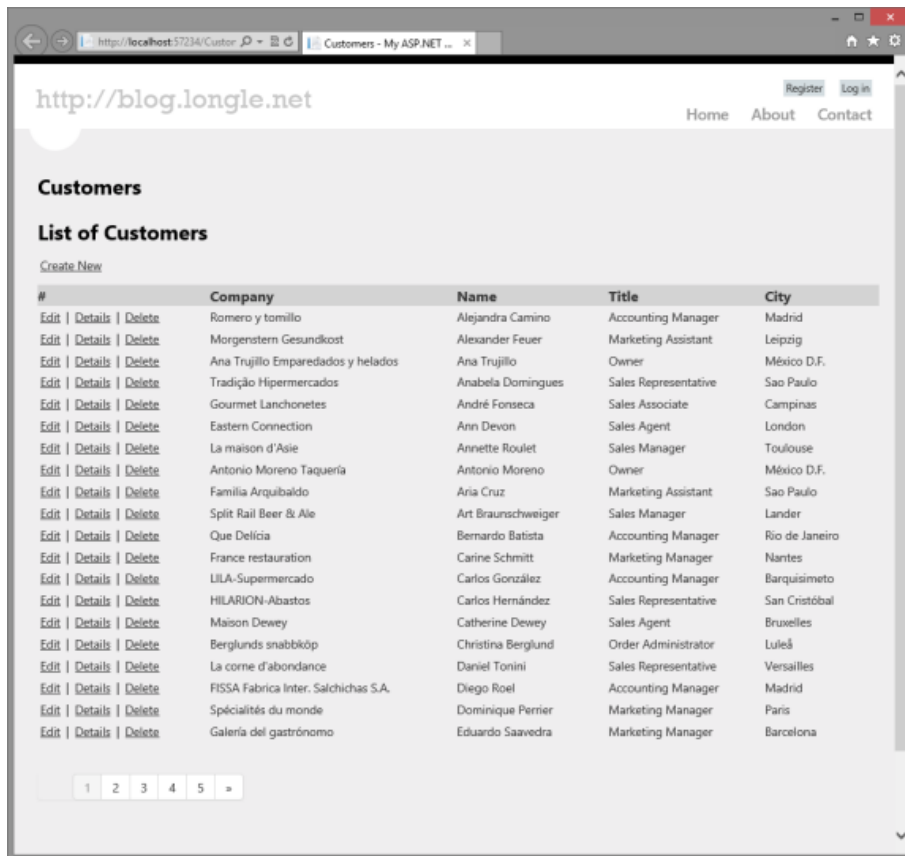
```

```

8 <h2>Customers</h2>
9
10 <link href="/Content/PagedList.css" rel="stylesheet" type="text/css" />
11
12 <h2>List of Customers</h2>
13
14 <p>
15     @Html.ActionLink("Create New", "Create")
16 </p>
17 <table style="width: 100%; padding: 10px;">
18     <tr style="background-color: lightgray; padding: 10px;">
19         <th>#</th>
20         <th>Company</th>
21         <th>Name</th>
22         <th>Title</th>
23         <th>Order Date</th>
24     </tr>
25
26     @foreach (var item in ViewBag.Customers)
27     {
28         <tr>
29             <td>
30                 @Html.ActionLink(
31                     "Edit", "Edit", new { id = item.CustomerID }) |
32
33                 @Html.ActionLink(
34                     "Details", "Details", new { id = item.CustomerID }) |
35
36                 @Html.ActionLink(
37                     "Delete", "Delete", new { id = item.CustomerID })
38             </td>
39             <td>@item.CompanyName</td>
40             <td>@item.ContactName</td>
41             <td>@item.ContactTitle</td>
42             <td>
43                 @if (item.Orders.Count > 1)
44                 {
45                     @item.Orders[1].OrderDate.ToShortDateString()
46                 }
47             </td>
48         </tr>
49     }
50
51     <tr>
52         <td colspan="9">
53             @Html.PagedListPager(
54                 (IPagedList)ViewBag.Customers, page =>
55                 Url.Action("Index", new { page }))
56         </td>
57     </tr>
58 </table>
59
60
61
62
63
64
65
66
67
68
69
70

```

Go ahead and run our project to see our paged customers grid.

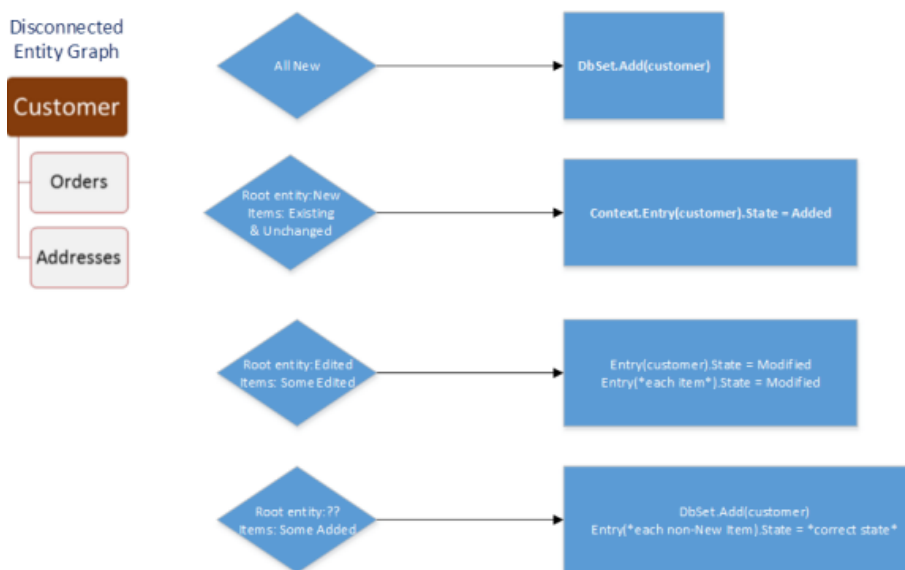


([http://lelong37.files.wordpress.com/2013/05/5-](http://lelong37.files.wordpress.com/2013/05/5-10-2013-6-46-30-pm1.png)

[10-2013-6-46-30-pm1.png](http://lelong37.files.wordpress.com/2013/05/5-10-2013-6-46-30-pm1.png))

Abstracting the Complexity when Dealing with Entity Graphs

Another item I wanted to go over was insert and updating graphs with our Repository pattern. There are four use cases for inserting graphs, they are as follows.



([http://lelong37.files.wordpress.com/2013/05/5-](http://lelong37.files.wordpress.com/2013/05/5-10-2013-7-34-28-pm.png)

[10-2013-7-34-28-pm.png](http://lelong37.files.wordpress.com/2013/05/5-10-2013-7-34-28-pm.png))

To abstract the complexity and EF experience required, and how the DbContext manages graphs e.g to know to set the root entity state and how it affects other entities in the graph (e.g. updating the root entity in the graph, and existing entities in the graph are to be updated or deleted) we added a interface IObjectState that all of our entities will implement.

Entities.IObjectState.cs

```
1 namespace Data
2 {
3     public interface IObjectState
4     {
5         ObjectState State { get; set; }
6     }
7 }
```

Entities.ObjectState.cs (enum)

```

1 namespace Data
2 {
3     public enum ObjectState
4     {
5         Unchanged,
6         Added,
7         Modified,
8         Deleted
9     }
10 }

```

These two classes will allow our development team to explicitly set the state of each of the entities in the graph when inserting or updating a graph. To make use of the classes we'll need to extend the DbContext with a few methods, we'll do this by creating extension methods.

Data.DbContextExtension.cs

```

1 public static class DbContextExtension
2 {
3     public static void ApplyStateChanges(this DbContext dbContext)
4     {
5         foreach (var dbEntityEntry in dbContext.ChangeTracker.Entries())
6         {
7             var entityState = dbEntityEntry.Entity as IObjectState;
8             if (entityState == null)
9                 throw new InvalidCastException(
10                     "All entites must implement " +
11                     "the IObjectState interface, this interface " +
12                     "must be implemented so each entites state" +
13                     "can explicitly determined when updating graphs.");
14
15             dbEntityEntry.State = ConvertState(entityState.State);
16         }
17     }
18
19     private static EntityState ConvertState(ObjectState state)
20     {
21         switch (state)
22         {
23             case ObjectState.Added:
24                 return EntityState.Added;
25             case ObjectState.Modified:
26                 return EntityState.Modified;
27             case ObjectState.Deleted:
28                 return EntityState.Deleted;
29             default:
30                 return EntityState.Unchanged;
31         }
32     }
33 }

```

Now we will override the SaveChanges in our NorthwindContext to invoke the ApplyStateChanges method to synchronize our ObjectState with EF's EntityState, so that the context will know how to deal with each and every entity when dealing with entity graphs.

Data.NorthwindContext.SaveChanges()

```

1 public override int SaveChanges()
2 {
3     this.ApplyStateChanges();
4     return base.SaveChanges();
5 }

```

Now when inserting, updating you can explicitly set the entities state, especially useful when dealing with graphs. This abstracts the skill-set of a developer using our Repository of having to know the what, when and how to set the state of entities in the graph in order for the context to update the graph and persist the graph correctly. Let's take a look at an example of updating an existing Order and adding an OrderDetail item with an entity graph. Both these actions, are will be executed on the same graph, however notice that the action is different for both of the entity's, one is updating and the other is adding, however we will only be invoking one method (IRepository.Update(TEntity entity)) from our IRepository in one transaction.

So we'll demonstrate and prove out updating an entity graph with our UnitOfWork implementation in these steps.

Code Snippet from [LinqPad \(http://www.linqpad.net/\)](http://www.linqpad.net/), notice how we are explicitly setting each of the entities state in the entity graph.

```

1 var unitOfWork = new UnitOfWork(this);

```

```

2
3  var orderId = 10253;
4
5  unitOfWork
6      .Repository<Order>()
7      .Query()
8      .Include(t => t.Customer)
9      .Filter(t => t.OrderID == orderId)
10     .Get().Dump();
11
12  var order = unitOfWork
13      .Repository<Order>()
14      .FindById(orderId);
15
16  unitOfWork
17      .Repository<OrderDetail>()
18      .Query()
19      .Filter(t => t.OrderID == orderId)
20      .Get().Dump();
21
22  order.ShipName = "Long Le";
23  order.State = ObjectState.Modified;
24
25  order.OrderDetails.Add(
26      new OrderDetail{
27          ProductID = 2,
28          UnitPrice = 10,
29          Quantity = 2,
30          Discount = 1,
31          State = ObjectState.Added
32      }
33  );
34
35
36  unitOfWork.Repository<Order>()
37      .Update(order);
38
39  unitOfWork.Save();
40
41  new UnitOfWork(this)
42      .Repository<Order>()
43      .Query()
44      .Include(t => t.Customer)
45      .Filter(t => t.OrderID == orderId)
46      .Get().Dump();
47
48  new UnitOfWork(this)
49      .Repository<OrderDetail>()
50      .Query()
51      .Filter(t => t.OrderID == orderId)
52      .Get().Dump();
53

```

Entity Graph Update Scenario

1. Query the Order table, make note of the ShipName value.
2. Query the OrderDetail table, make note there are only three (3) items, that belong to the same Order.
3. Update the ShipName value in the Order.
4. Add an OrderDetail to the Order.

| OrderID | CustomerID | EmployeeID | OrderDate | RequiredDate | ShippedDate | ShipVia | Freight | ShipName | ShipAddress | ShipCity | ShipRegion | ShipPostalCode | ShipCountry | State |
|---------|------------|------------|-----------------------|-----------------------|-----------------------|---------|---------|--------------|-----------------|----------------|------------|----------------|-------------|-----------|
| 10253 | HANAI | 3 | 7/10/1996 12:00:00 AM | 7/24/1996 12:00:00 AM | 7/16/1996 12:00:00 AM | 2 | 58.1700 | Hanan Carnes | Rua do Paço, 67 | Rio de Janeiro | RJ | 05454-876 | Brazil | Unchanged |

| OrderID | CustomerID | EmployeeID | OrderDate | RequiredDate | ShippedDate | ShipVia | Freight | ShipName | ShipAddress | ShipCity | ShipRegion | ShipPostalCode | ShipCountry | State | Product | State |
|---------|------------|------------|-----------------------|-----------------------|-----------------------|---------|---------|----------|-----------------|----------------|------------|----------------|-------------|----------|---------|-------|
| 10253 | HANAI | 3 | 7/10/1996 12:00:00 AM | 7/24/1996 12:00:00 AM | 7/16/1996 12:00:00 AM | 2 | 58.1700 | Long Le | Rua do Paço, 67 | Rio de Janeiro | RJ | 05454-876 | Brazil | Modified | | |

(<http://lelong37.files.wordpress.com/2013/05/5-11-2013-4-13-23-pm.png>)

(click image to enlarge)

Presto, we were able to successfully update an existing Order and add a new OrderDetail via an entity graph with one transaction using one method. Now, we can absolutely do this using EF out of the box, however, our goal here is was to abstract the complexity and skill set required from a developer in regards to EF specially how do deal with the DbContext in order to make this happen as well as obviously still support working with graphs through our IRepository implementation.

There you have it, and extensible genericized implementation of the UoW and Repository pattern with EF in MVC. In the [next blog post](http://blog.longle.net/2013/05/17/generically-implementing-the-unit-of-work-repository-pattern-with-entity-framework-in-mvc-simplifying-entity-graphs-part-2-mvc-4-di-ioc-with-mef-attributeless-conventions/) (<http://blog.longle.net/2013/05/17/generically-implementing-the-unit-of-work-repository-pattern-with-entity-framework-in-mvc-simplifying-entity-graphs-part-2-mvc-4-di-ioc-with-mef-attributeless-conventions/>), we'll add DI & IoC to this solution and introduce the Services layer, this layer will house all of our business logic and rules. We will also implement the Services layer in a way, where we don't violate our Unit of Work pattern, meaning all the work done in our Repository and Services are executed under one single instance the DbContext per page request.

Happy Coding..! 😊

Download sample: <https://genericunitofworkandrepositories.codeplex.com> (<https://genericunitofworkandrepositories.codeplex.com>)

About these ads (<http://en.wordpress.com/about-these-ads/>)

Posted by [Le](#) in [.NET](#), [EF](#), [MVC 3](#), [MVC 4](#), [Uncategorized](#)

Tagged: [net](#), [architecture](#), [best practice](#), [C#](#), [ef](#), [entity framework](#), [extendable](#), [extensible](#), [framework](#), [generic](#), [graph](#), [mvc](#), [pattern](#), [repositories](#), [repository](#), [reusable](#), [unit of work](#)

112 thoughts on “Generically Implementing the Unit of Work & Repository Pattern with Entity Framework in MVC & Simplifying Entity Graphs”

1. [Sam](#) says:

May 15, 2013 at UTC

Reblogged this on [Sailing The Sahara](#) and commented:

Looking forward to another article from Long at <http://wordpress.com/#!/read/blog/id/10143612/>

Can I link to him that way? Oh well, I'm sure I'll be flamed for offending the blog gods and learn from it!

I'm busy working on a few very small Asp.net MVC projects that I will be public soon. I look forward to sharing.

Reply

◦ **Le** says:

May 31, 2013 at UTC

Sam, thx for the ReBlog!

2. Pingback: [*Generically Implementing the Unit of Work & Repository Pattern with Entity Framework in MVC & Simplifying Entity Graphs – Part 2 \(MVC 4 & DI, IoC with MEF Attributeless Conventions\) | Long Le's Blog*](#)
3. Pingback: [*Generically Implementing the Unit of Work & Repository Pattern with Entity Framework in MVC & Simplifying Entity Graphs – Part 2 \(MVC 4 & DI, IoC with MEF 2 Attributeless Conventions .NET 4.5\) | Long Le's Blog*](#)
4. Pingback: [*MVC 4 with DI & IoC with MEF 2 Attribute-less using Conventions with RegistrationBuilder in .NET 4.5 – Part 2 | Long Le's Blog*](#)
5. Pingback: [*Implementing DI & IoC Pattern in MVC 4 with MEF 2 Attributeless using Conventions with RegistrationBuilder in .NET 4.5 – Part 2 | Long Le's Blog*](#)

◦ **Le** says:

May 20, 2013 at UTC

Here is the latest download url, please let me know if there are any issues with it.

<https://skydrive.live.com/redir?resid=949A1C97C2A17906!5107&authkey=!AIfx5T8CZ5LlxAs>

6. **cmarty83** says:

May 21, 2013 at UTC

Very good article but the Query method should return an interface IRepositoryQuery instead of a concrete object to be independent with Entity Framework.

Reply

◦ **Le** says:

May 21, 2013 at UTC

Hi cmarty83, aaah, good point will update the post and solution..

7. **Kristof Heiremans** says:

May 29, 2013 at UTC

Hi,

i've implemented your Repo/UoW pattern but I'm facing problems when trying to eager load some entities.

This simply doesn't seem to be working unless the entities are already loaded by the UoW (in a different call).

Have you experienced any of these problems?

thanks for your help

Reply

◦ **Kristof Heiremans** says:

May 29, 2013 at UTC

Just to inform.

I've also tried eager loading against the DbContext without using the Includes from RepositoryQuery and then everything is working smoothly.

Could you check at your end if your code is performing ok?

◦ **Le** says:

May 29, 2013 at UTC

Kristof, good catch! I forgot to append the include, I've updated the download link in the post with the fix.

Before:

```
1 | if (includeProperties != null)
2 |     includeProperties.ForEach(i => query.Include(i));
```

After:

```
1 | if (includeProperties != null)
2 |     includeProperties.ForEach(i =>{ query = query.Include(i); });
```

Last but not least I've disabled lazy loading for best practices, this is to help people from accidentally loading large entity graphs.

Data.NorthwindContext

```

1 | public NorthwindContext()
2 |     : base("Name=NorthwindContext")
3 | {
4 |     Configuration.LazyLoadingEnabled = false;
5 | }
```

8. **medillogicchris** says:

May 31, 2013 at UTC

Is it possible to do joins using the repository system?

Thanks

Chris

Reply

◦ **Le** says:

May 31, 2013 at UTC

The IUnitOfWork.Repository() will return the TEntity along with everything that it is associated with (complex member types), just make sure you don't forget to eager load them. If there is a join you need to do that is beyond this, then you can simply write an extension method for static GetCustomerCustomQuery(IRepository this, param1, param2), extending the IRepository is covered [here](#) in the post #Extensibility.

◦ **medillogicchris** says:

May 31, 2013 at UTC

Thanks Le. Excellent article

◦ **Le** says:

May 31, 2013 at UTC

thx for the positive feedback

9. **Brian** says:

June 10, 2013 at UTC

How did you get this to work with LinqPad? When executing a query, I'm getting the following error:

Unable to cast object of type 'LINQPad.LINQPadDbConnection' to type 'System.Data.SqlClient.SqlConnection'.

Reply

◦ **Le** says:

June 11, 2013 at UTC

You will need to NuGet EF 6 and add all the required references (DLL's). I simply referenced all of them from the debug directory of the Web project.

1. Data.dll
2. Entities.dll
3. EntityFramework.dll
4. Repository.dll

You will also need to add a LinqPad connection using a "Typed data context from you own assembly" with Entity Framework (DbContext v4.1 – to v6.0) selected as well as browsing the the web.config so that LinqPad is aware of the connection string to use. This question has been asked a few times, so mostly will be my next blog post, I'll tweet the post when this it's published, if you want you can follow me on <http://twitter.com/lelong37> for the update.

◦ **Brian** says:

June 11, 2013 at UTC

Thanks for responding. Still no luck as I continue to get the same error. As far as I can tell, everything is setup as described in your article. The only difference is that I'm using EF6-Beta1.

◦ **Le** says:

June 11, 2013 at UTC

Here's the post on how query using the Unit of Work and Repository implementation with LinqPad

<http://blog.longle.net/2013/06/11/using-linqpad-to-query-and-troubleshoot-custom-entity-framework-dbcontext-unit-of-work-and-repository-pattern/>

◦ **Le** says:

June 12, 2013 at UTC

Brian,

Where you able to get LinqPad working?

◦ **Brian** says:

June 12, 2013 at UTC

Yes, with EF5. No luck with EF6-Beta1.

10. Pingback: [*Using LinqPad to Query & Troubleshoot Custom Entity Framework DbContext, Unit of Work & Repository Pattern | Long Le's Blog*](#)

11. Pingback: [*Modern Web Application Layered High Level Architecture with SPA, MVC, Web API, EF | Long Le's Blog*](#)

12. **Bruce** says:

June 17, 2013 at UTC

I'm a bit confused about something. In your repository you reference the IDbSet interface which requires you to reference Entity Framework. I thought the goal of this was to remove your dependence from Entity Framework within your repository. Am I missing something?

Reply

◦ **Le** says:

June 17, 2013 at UTC

Hi Bruce,

The goal was to remove the dependencies from EF from anything above the repository layer, which in our case could be apps or (web) services. If you were ever to change out your ORM e.g. EF, you would only have to deal with changes in the Repository layer (project).

13. **Richard Cullen** says:

June 21, 2013 at UTC

Hi Le

I was looking for a generic implementation of the Unit of Work and Repository patterns when I came across your blog. It has helped me immensely and I appreciate that. In the initial blog about implementing the patterns you talked about extending the IRepository interface for specific objects.

I'm trying to implement that and had assumed that I should call the extension method in this form:

```
IUnitOfWork.Repository().ExtensionMethod();
```

However Visual Studio complains that IRepository does not contain a definition for 'ExtensionMethod' and no extension method 'ExtensionMethod' accepting a first argument of type IRepository could be found.

My IRepository interface, implementation and the static class containing the extension methods all share the same namespace so it shouldn't be a reference issue. My static class is defined as:

```
public static class ExtensionClass
{
    public static SpecificEntity GetSingleEntity(this IRepository seRepository, int id)
    {
    }
}
```

Am I calling the extension method in the correct manner? Do you see any other potential issue in the code?

Reply

◦ **Richard Cullen** says:

June 21, 2013 at UTC

Oops, apparently the copy and paste from the email upset a few things: The call I'm making is

```
IUnitOfWork.Repository().ExtensionMethod();
```

and the function is

```
public static SpecificEntity GetSingleEntity(this IRepository seRepository, int id)
```

Thanks

◦ **Richard Cullen** says:

June 21, 2013 at UTC

Ok, sorry for the chain of comments. It's WordPress seeing my template declarations as a tag. Guess I should have put it in a code section

Call:

```
IUnitOfWork.Repository().ExtensionMethod(id);
```

Function:

```
public static SpecificEntity GetSingleEntity(this IRepository seRepository, int id);
```

o **Le says:**

June 21, 2013 at UTC

Hi Richard,

It's best practice to have the extension method this way:

(example)

```
1 public static class CustomerRepository
2 {
3     public static void AddCustomerWithAddressValidation(
4         this IRepository<Customer> customerRepository, Customer customer)
5     {
6         USPSManager m = new USPSManager("YOUR_USER_ID", true);
7         Address a = new Address();
8         a.Address1 = customer.Address;
9         a.City = customer.City;
10
11         Address validatedAddress = m.ValidateAddress(a);
12
13         if (validatedAddress != null)
14             customerRepository.InsertGraph(customer);
15     }
16 }
```

Notice in the example the extension method is extending (first parameter)

```
1 | this IRepository<Customer> customerRepository
```

Reason is, in the UnitOfWork implementation we are actually returning IRepository<T> .

```
1 public IRepository<T> Repository<T>() where T : class
2 {
3     if (_repositories == null)
4         _repositories = new Hashtable();
5
6     var type = typeof (T).Name;
7
8     if (!_repositories.ContainsKey(type))
9     {
10         var repositoryType = typeof (Repository<>);
11
12         var repositoryInstance =
13             Activator.CreateInstance(repositoryType
14                 .MakeGenericType(typeof (T)), _context);
15
16         _repositories.Add(type, repositoryInstance);
17     }
18
19     return (IRepository<T>) _repositories[type];
20 }
```

Also, could you please confirm that in the class that is calling your new extension method that it indeed has a using statement with the fully qualified namespace for both the standard generic repository and the fully qualified namespace for newly added extension method? In the download sample app, these happen to be in the same namespace.

4. **Vish says:**

June 23, 2013 at UTC

I stumbled across this blog while looking for MEF, EF, UOW and how they interact. This blog is a treasure trove of data. Fantastic job on putting it together Le!

Reply

o **Le says:**

June 23, 2013 at UTC

Thanks for the positive feedback Vish...!

[5. **manjerekar rao** says:

July 6, 2013 at UTC

hi

i'm using the db first approach for EF, This is how my Context class is. its auto generated and i cannot modify it. coz it will get re-generated even if i modify it.

i'm getting error in the Repository class at this location....pls help, i liked your tutorial, but i'm stuck now

```
public class Repository : IRepository where TEntity : class
```

```
{
    internal IDbContext Context;
    internal IDbSet DbSet;
```

```
    some code to follow.....
}
```

```
public partial class ORPEntities : DbContext
```

```
{
    public ORPEntities()
        : base("name=ORPEntities")
    {
    }
}
```

```
protected override void OnModelCreating(DbModelBuilder modelBuilder)
```

```
{
    throw new UnintentionalCodeFirstException();
}
```

```
public DbSet Addresses { get; set; }
public DbSet DataDictionaries { get; set; }
public DbSet Documents { get; set; }
public DbSet EmployerDetails { get; set; }
public DbSet EmployerInEventsInJobs { get; set; }
public DbSet Events { get; set; }
public DbSet Jobs { get; set; }
public DbSet Memberships { get; set; }
public DbSet OAuthMemberships { get; set; }
public DbSet Roles { get; set; }
public DbSet SelectionCodes { get; set; }
public DbSet SelectionProcesses { get; set; }
public DbSet sysdiagrams { get; set; }
public DbSet TaskManagers { get; set; }
public DbSet UserEducationDetails { get; set; }
public DbSet UserInJobs { get; set; }
public DbSet UserITSkills { get; set; }
public DbSet UserPersonalDetails { get; set; }
public DbSet UserProfiles { get; set; }
public DbSet UserResults { get; set; }
public DbSet UsersInRoles { get; set; }
public DbSet UsersInTasks { get; set; }
public DbSet UserWorkExperiences { get; set; }
}
```

Reply◦ **Le** says:

July 7, 2013 at UTC

Could you please include the actual error?

[6. **manjerekar rao** says:

July 7, 2013 at UTC

```
internal IDbContext Context;
internal IDbSet DbSet;
```

assembly not found for both the interfaces.

to avoid this error i need to copy your interfaces which is in your Data layer.

But the actual change here is that i have to modify my context class to implement the interface.

```
public class NorthwindContext : DbContext, IDbContext
{
}
}
```

since you are using the code first approach its fine to modify it. but in my case the NorthwindContext is automatically created by using db first approach and it extends DbContext only and any change to this NorthwindContext will be automatically deleted when i run the command update model from database which is provided by the EF.

i hope its clear for you.

thank you

Reply

◦ **Le** says:

July 7, 2013 at UTC

Can you manually re-apply your DbContext over the generated one after your update? or you just create a partial class for the NorthwindContext?, or create another abstract custom DbContext (NorthwindContextBase) which would inherit DbContext e.g. NorthwindContext -> NorthwindContextBase (abstract) -> DbContext (abstract)?

◦ **Le** says:

July 7, 2013 at UTC

BTW, why don't you just simply delete the generated one? The DataContext in this post is "Generic" meaning you never have to update it, no matter how many new models are added, deleted or modified, because you access them by Repository<Customer>(), meaning, just ignore the generated one, you don't even need it.

17. **Tim** says:

July 25, 2013 at UTC

Hi Long,

I keep coming back to your blog from google. And I must say, it's one of the easier to understand explanations of setting up repositories and units of work, etc that I've come across. I'm an 8th grade teacher here in Southern California, and trying to learn programming over the summer to build a better way for students to learn. Question I have is I want to use bounded contexts as my database will grow with more tables, and it makes sense to me. Julie Lerman suggests using using a separate unit of work for each dbContext. But when setting up the concrete units of works in IOC, how would structuremap know which concrete unit of work to use? I can have IUnitOfWork uow and the whole point is that structuremap knows which concrete instance to pick up — easy if there's just one, but how would structuremap know which one to inject in an mvc4 application when you potentially have 5 different dbcontexts, etc. You'd think it'd be confused, and I really want to use bounded contexts, and IOC at the same time. Thank you,

Tim

Reply

◦ **Le** says:

July 30, 2013 at UTC

Hi Time,

Didn't get a chance to start working on this until tonight, anyhow here you go.

<http://blog.longle.net/2013/07/30/bounded-dbcontext-with-generic-unit-of-work-generic-repositories-entity-framework-6-unity-3-0-in-mvc-4/>

Kudos, for the great question.

18. **Jonathan** says:

August 3, 2013 at UTC

Hello!

This is a fantastic post. I am glad I stumbled across it. I have previously implemented solutions using SharpArchitecture, however, I wanted to get away from the reliance on its framework in order to get a better understanding of the Repository pattern and UoW in an MVC3/4 solution.

I downloaded your sample solution (linked by this post) and am attempting to use your bits with Castle.Windsor, although seeing how easy it is implementing Unity now, I might switch.

I have two questions.

Firstly, I didn't see anywhere in your sample solution where the DbContext was being created and disposed in the lifetime of a Request. Is this handled internally in Unity? Am I missing something obvious? I've been accustomed to using NHibernate, so EF is a bit new for me.

Secondly, I am curious about some of the references in each assembly. I noticed the Repository assembly references several System.Web assemblies, but it appears to only be relying on System.Web.Mvc which looks like it points to the DependencyResolver in the Bootstrapper. Would it be better if the Bootstrapper implementation lived in the web project (Spa)?

Thanks again for a great series of articles!

Reply

◦ **Le** says:

August 3, 2013 at UTC

Thx, for the positive feedback Jonathan.

Unity v3.0 is awesome, I'd make the switch

Answer 1:

If you view this post: [Bounded DbContext with Generic Unit of Work, Generic Repositories, Entity Framework 6 & EntLib Unity 3.0 in MVC 4](#) and scroll down to the section titled: Spa.App_Start.UnityWebActivator.cs, this is where we default everything's lifecycle to the lifecycle of the request.

Answer 2:

I'm working on uploading v2.0 of this solution that has optimizations to DOM management (letting Kendo manage all of this), View caching improvements, and uses WebActivator vs. BootStrapper (which I thought this solution was already using), and much more ASP.NET MVC developer friendly (based on MVC dev community feedback). Hoping to upload it before the weekend is over, I'll update the blog series when its uploaded.

!9. **Daskul** says:

August 6, 2013 at UTC

Hi this is an awesome post. very clean! not like other articles that extending IRepository and creating concrete implementation which is very time consuming and you have to touch the UnitOfWork everytime you have to add a repository because Repositories are members unlike with your design that they are automagically created.

I just have a question because youve mention "to abstract away ORM tool like EF" but in your IDbContext you have a method that returns IDbSet which is under the EF assembly. what if I use other ORM? can I implement that method?

Reply

◦ **Le** says:

August 7, 2013 at UTC

Thanks Daskul for the positive feedback. Yes there was some deep thought around "Open and Close" principle when designing and implementing the generic Unit of Work and Repository. In theory, yes you should be able implement this generic Unit of Work and Repository Pattern with any ORM, however when doing so there maybe a few very minor changes, e.g. wrapping the other ORM's version of DbSet.

◦ **daskul** says:

August 7, 2013 at UTC

thank you for your quick response.

does it mean I still have to reference EF when using other ORM?

and I have another question, where should the Extension Methods live? in the Repository assembly? and isnt it RepositoryQuery is just like exposing an IQueryable on client? what is the difference between the two? why not just provide extension methods and no more query?

!0. **HorseKing** says:

August 7, 2013 at UTC

Hi buddy, I wonder how I can use OrderByDec?? thanks

Reply

!1. **HorseKing** says:

August 7, 2013 at UTC

ah ha, never mind, I figure it out.. after call the first OrderBy, immediately call OrderByDescending.

Reply

!2. **A.T** says:

August 12, 2013 at UTC

Hi Le,

Thanks for the great post.

It is really one of the best

I'm using this framework for my project in which we have more than 100 tables. for small proof of concept with let's say 20 tables it worked perfectly fine though when I wanted to just scale this up for the big database which effectively have big object graphs I got the Out of Memory exception when using following pattern from my controller

```
UnitOfWork.Repository.Query().Get()
```

However

```
UnitOfWork.Repository.FindById(id) works perfectly fine .
```

I'm using EF 5 with .NET 4.5 VS 2012

Am I missing some part ?

For simplicity following is the code

on my client app

```
public MyPOCOClass GetByNumber(string myClassNumber)
{
    IEnumerable mylist ;
    MyPOCOClass item ;
    using (UnitOfWork uow = new UnitOfWork())
    {
        IRepository repo = uow.Repository();

        mylist = repo.All;

        repo.FindById(2023182); //Works fine

        var itemlist = from i in mylist
            where (i.ClassNumber==myClassNumber)
            select i ;
        item = itemlist.FirstOrDefault();
    }

    return item;
}
```

On Generic Repository:

```
public IQueryable All
{
    get { return Context.Set(); }
}
```

And on Unit of Work:

```
public IRepository Repository() where T : class
{
    //Repository r = new Repository(_context);
    //return (IRepository)r;

    var repositoryType = typeof(Repository);

    var repositoryInstance =
        Activator.CreateInstance(repositoryType
            .MakeGenericType(typeof(T)), _context);

    return (IRepository)repositoryInstance;
}
```

I have exposed the IDbSet through All property as I thought I may have missed something to do with Repository hash table or Query method or Get method and simply I'm not using them to locate the issue though no chance

Here is The Exception:

An exception of type 'System.OutOfMemoryException' occurred in System.Data.Entity.dll but was not handled in user code

Stack Trace:

```
at System.Collections.Generic.Dictionary`2.Resize(Int32 newSize, Boolean forceNewHashCodes)
at System.Collections.Generic.Dictionary`2.Insert(TKey key, TValue value, Boolean add)
at System.Data.Objects.ObjectStateManager.AddEntryToKeylessStore(EntityEntry entry)
at System.Data.Objects.ObjectStateManager.AddEntityEntryToDictionary(EntityEntry entry, EntityState state)
at System.Data.Objects.ObjectStateManager.AddEntry(IEntityWrapper wrappedObject, EntityKey passedKey, EntitySet entitySet, String argumentName, Boolean isAdded)
at System.Data.Common.Internal.Materialization.Shaper.HandleEntityAppendOnly[TEntity](Func`2 constructEntityDelegate, EntityKey entityKey, EntitySet entitySet)
at lambda_method(Closure , Shaper )
at System.Data.Common.Internal.Materialization.Coordinator`1.ReadNextElement(Shaper shaper)
at System.Data.Common.Internal.Materialization.Shaper`1.SimpleEnumerator.MoveNext()
at System.Linq.Enumerable.WhereEnumerableIterator`1.MoveNext()
at System.Linq.Enumerable.FirstOrDefault[TSource](IEnumerable`1 source)
at
```

which is happening on GetByNumber(string myClassNumber)

I know that this could be confusing, though I really appreciate it if you can share some thoughts or guideline

Cheers,

Reply

◦ **Le** says:

August 12, 2013 at UTC

A.T, hmmm, strange, quite a few teams are using this in production with fairly large databases (tables) with a lot data throughput. Are you using in DI & IoC, if so, may be an issue with the lifetime management, if you are, which framework are you using? If not, is the UnitOfWork have singleton like behavior? Also, what kind of app is this in (e.g. MVC, WPF, WCF)?

13. **Jean** says:

August 14, 2013 at UTC

Hey, this is a great post i have learned a lot. I have a question i have an object that has a child list of object and when i send them to update to the repository im recieving a constration error like this:

```
{ "An object with the same key already exists in the ObjectStateManager. The ObjectStateManager cannot track multiple objects with the same key." }
```

How can i fix this problem when updating entities with child items.

Regards,

Reply

◦ **Le** says:

August 14, 2013 at UTC

Did you set the entity.State for every entity in the graph?

14. **A.T** says:

August 14, 2013 at UTC

Hi Guys,

As I've got it cracked just wanted to give you a head around the reason I was facing the issue

In the following code even though All property is an IQueryable I was getting it in an IEnumerable mylist which make the Entityframework build the object graph for 2 Million records which is causing the An exception of type 'System.OutOfMemoryException'.

The work around was to get the IQueryable and filter the result set on service layer or get the RepositoryQuery return IEnumerable just in case the reult set is filtered.

```
public MyPOCOClass GetByNumber(string myClassNumber)
{
```

```
    IEnumerable mylist ; // wrong leave it as var which will return IQueryable
```

```
    MyPOCOClass item ;
```

```
    using (UnitOfWork uow = new UnitOfWork())
```

```
    {
        IRepository repo = uow.Repository();
```

```
mylist = repo.All;

repo.FindById(2023182); //Works fine

var itemlist = from i in mylist
where (i.ClassNumber==myClassNumber)
select i ;
item = itemlist.FirstOrDefault();
}

return item;
}
```

On Generic Repository:

```
public IQueryable All
{
get { return Context.Set(); }
}
```

Reply

- **Le** says:
August 15, 2013 at UTC
Great, glad you found a fix for it!

15. **Carlos** says:
August 16, 2013 at UTC
Hello Le,

Very good article!

I've got a question for you:

Have you got some generators for the entities classes? Because if you've 40/50/80 tables...and you need to have all the classes about this tables in your solution, do you use some generator for the entities? and mapping?

Keep up with good job!

Regards.
CM

Reply

- **Le** says:
August 16, 2013 at UTC
Thanks. Yes, I use Entity Framework Power Tools, from the Entity Framework Team
(<http://visualstudiogallery.msdn.microsoft.com/72a60b14-1581-4b9b-89f2-846072eff19d>).

- **Carlos** says:
August 16, 2013 at UTC
Thank you so much for quick answer!

Last question (lol):

I don't know if you use stored procedures in your solutions, but in case that you have to call some stored procedures, what could be the best practice for that?

Regards,
CM

- **Le** says:
August 16, 2013 at UTC
You can extend the UnitOfWork add add a method StoredProc<T>, the type that you pass in could be the Model that EF generates for the StoredProc when using EF Power Tools. EF Power Tools will generate Tables, Views, SPROCS, etc.
- **Carlos** says:
August 16, 2013 at UTC
Thanks!!!
- **Matt Ford** says:
October 7, 2013 at UTC

Hi Le,

Again I'd like to echo your work at bringing it all together so nicely. I've already implemented most of this work, but am getting slightly unstuck when it comes to writing in the Stored Procedure part in. Could you elaborate on an example please? I just don't know how to get my EF UoW to run the SP and return in the complex object type that I've got via using DatabaseFirst.

```
private readonly IDbContext _context;
public IList StoredProcedure(object[] parameters) where TModel : class
{
    return null;
}
```

Any help?

Thanks.

- **Le** says:
October 8, 2013 at UTC
Hi Matt,

Thanks for the positive feedback, we are releasing v2.0 of this framework this weekend, it will support sprocs.

For now you can see it @ :

<https://genericunitofworkandrepositories.codeplex.com/SourceControl/latest#main/Repository/Repository.cs>

```
1 | public virtual IQueryable<TEntity> SqlQuery(string query, params object[] parameters)
2 | {
3 |     return _dbSet.SqlQuery(query, parameters).AsQueryable();
4 | }
```

You would call it this way:

```
1 | _unitOfWork.Repository<Customer>().SqlQuery("sp_MyStoredProc @firstName, @lastName", param:
```

If you want you can go ahead and download what's in the main branch, it's in beta though.

- **Matt Ford** says:
October 7, 2013 at UTC
Lol that should read "echo the great comments regarding your work"
- **Matt Ford** says:
October 8, 2013 at UTC
Hi Le,

Thanks for the speedy reply. The solution you've posted is what I've got already and I feel a little unsettled at exposing EF to SQL input by any developer. The solution I've coded is very generically so that I am exposing via interfaces and base classes only what I want them to use. I have the repository pattern and unit of work going on, but also have a layer in between the business logic and EDMX which takes in the unit of work and sets up a new repository and context for each POCO. It means that one unit of work is getting thrown about in the business logic and recording loads of changes instead of just one graphs changes.

Guide:

<http://stackoverflow.com/questions/15181290/decouple-unit-of-work-from-services-or-repo/15527444#15527444>

However, this has led me to my stored procedure issue. I want to be able to import my SP into DatabaseFirst, create my DAL-ish layer and call the SP. Normally I (literally) call Repository.GetList as I pass in the type generically and it handles the rest, so I want something just as simple.

Was that a load of waffle or does that make sense?

Thanks

- **Le** says:
October 10, 2013 at UTC
Hmm, interesting, haven't had a use case for this particular scenario, however if you come up with an elegant solution, please do share.

- !6. **Andy Uhl** says:
August 21, 2013 at UTC

Hi Le. I'm a bit confused here and I wonder if you could help clarify. As you say above, "...our presentation layer...should not have any dependent code on EF assembly, therefore that should not be any references to the EF assembly, it should only reference our Repository project for data access."

However, I have downloaded your sample code and I see that the Spa project does in fact reference the Entity Framework library:

False

..\packages\EntityFramework.5.0.0\lib\net45\EntityFramework.dll

Can you clarify for me why that is and if I'm missing something or just confused?

Thanks,
Andy

Reply


◦ **Le** says:

August 21, 2013 at UTC

Good question, the assembly has been added as a reference so that it gets copied to the bin folder because the connection string for EntityFramework in the web.config requires the assembly for the sql provider. However, we don't have any actual code in the application that is dependent on EntityFramework. All data access happens through the IUnitOfWork and IRepository. Also, there is no EF dependencies in the Entity project as well, where are the POCO entities reside, which there shouldn't be, since they are actually plain old vanilla C# objects.

Connection String Example (providerName attribute requires the EntityFramework assembly):

```
1 <add name="NorthwindContext" connectionString="Data Source=(LocalDb)\v11.0;Initial Catalog=
2
```



You can create a folder named "Assemblies" add the EntityFramework.dll and the EntityFramework.SqlServer.dll to it, set the "Build Action" to "Content", set the "Copy to Output Directory" to "Copy Always" and remove these references from your project.

◦ **Andy Uhl** says:

August 22, 2013 at UTC

Thanks for the fast response, Le. But, I'm still having problems.

I went ahead and removed the EntityFramework reference from the Spa project. I also removed the AccountController and the other SimpleMembership files as they also reference EF directly and are not relevant to what I'm trying to do at the moment.

However, when building your solution (and mine, which is modeled after it), I receive the following error in the Bootstrapper:

The type 'System.Data.Entity.DbContext' is defined in an assembly that is not referenced. You must add a reference to assembly 'EntityFramework, Version=5.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089'.

This error is tied to line 25 which is the line where the NorthwindContext is registered, i.e., container.RegisterType();

The NorthwindContext object is derived from DbContext, which is in the EF library.

Thanks!
Andy

◦ **Le** says:

August 22, 2013 at UTC

Aaah, yes, Unity needs this for DI & IoC. We would have to remove the registration for IDbContext from UnityConfig.cs and move the registration to the web.config.

◦ **Le** says:

August 23, 2013 at UTC

Honestly, I wouldn't let that bother me, the reference is there only because Unity requires it for Registration for DI & IoC, there's no actual code in our application that uses Entity Framework directly for data access, which is what the objective is.

!7. **Andy Uhl** says:

August 23, 2013 at UTC

Thanks again for your responses. As you indicated, removing the reference and moving the configuration worked as you said it would. As you also touched on, whether or not we want to take that step for a very strict separation is now something we can decide on as either approach will work. It's nice to have the choice.

Your code has been very helpful!

Reply

- **Le** says:
August 24, 2013 at UTC
Thanks Andy, for the positive feedback

28. **rfreire** says:

September 5, 2013 at UTC

Would it be possible, somehow to support Select? I mean, just to select some columns/properties instead of the whole TEntity. Sometimes this is useful when dealing with complex types just to take the properties you need instead of the whole object graph.

Reply

- **Le** says:
September 12, 2013 at UTC
Hi rfreire, will add this to our backlog on <https://genericunitofworkandrepositories.codeplex.com/>, thx for the recommendation...!

29. **aadamxs** says:

September 12, 2013 at UTC

I just wanted to give feedback on my experience using the Entity ObjectState implementation above. I ran into an issue with Seeding the DB with EF (using the Configuration.Seed method). In simple terms, if you use the overridden SaveChange method that contains the this.ApplyStateChanges() method, you will run into issues with duplicate rows being added to the database.

The work around is found in this post on stackoverflow: <http://stackoverflow.com/questions/10007351/entity-framework-code-first-addorupdate-method-insert-duplicate-values/17913036#17913036>

I don't understand why this happens, and would really like to read an explanation. To solve I add another method called SaveSeedChanges() without the ApplyStateChanges — and it works fine.

Hope this helps!

Reply

- **Le** says:
September 12, 2013 at UTC
Appreciate the update aadamxs, this will be helpful for all of us that are doing code-first, including myself.

30. **Anders Persson** says:

September 16, 2013 at UTC

Hi.. If i want to Override FindById for a Entity in the UnitOfWork scope, how can i do that. I tried the following code, but this method will never be hit.

```
public class ProductRepository : Repository, IRepository
{
    public ProductRepository(IDbContext context)
    : base(context) { }

    public override Product FindById(object id)
    {
        return base.FindById(id);
    }
}
```

Reply31. Pingback: [Dew Drop – September 17, 2013 \(#1,625\) | Morning Dew](#)32. Pingback: [Lindermann's Blog | Implementação dos padrões UoW e Repository com Entity Framework](#)33. **Pavlos** says:

September 23, 2013 at UTC

Hi,

First of all i'd like to thank for this great post. Well, it seems that i have some problems running the above code.

I tried to test the above code; so i added a custom model CustomModel and inserted the following lines of code into my controller:

```
var uof = new UnitOfWork(new CustomContext());
var repo= uof.Repository();
repo.InsertGraph(new CustomModel());
uof.Save();
```

```
var list = repo.Query().Get().ToList()
```

The last line throw an exception:

Invalid object name 'dbo.CustomModel'

The error makes sense because in the "Server Explorer" in visual studio i cannot see any table named CustomModel

Can you tell what i'm doing wrong?

Cheers.

Reply

◦ **Le** says:

September 25, 2013 at UTC

You should only be using models that are from Entity Framework (POCO's). I'm not sure if CustomModel is a true Entity Framework POCO but it doesn't sound like it. You cannot just simply pass in any random C# object and expect the UoW and/or Repo framework and assume that it will know how to persist this to your database.

◦ **Jose Gregorio Taveras** says:

September 28, 2013 at UTC

If you are using migrations you have to make Update-Database or if you are using database first you need to generate the sql and execute it against the db

◦ **Pavlos** says:

October 2, 2013 at UTC

I do use migrations and i do make Update-Database but the problem is that when i then try to call WebSecurity.InitializeDatabaseConnection("connectionString", "UserProfile", ...) to create the WebMatric tables, it throws an exception saying that the userProfile table doesn't exist

34. **Angad Bhat** says:

September 23, 2013 at UTC

I am having an issue. When I insert an object say 'Customer' I get an error Invalid column name 'State'. I have set the state property to be added in my instance of customer object. When I don't add the state = added then the entity state remains unchanged and

Reply

◦ **Angad Bhat** says:

September 23, 2013 at UTC

I am having an issue. When I insert an object say 'Customer' I get an error Invalid column name 'State'. I have set the state property to be added in my instance of customer object. When I don't add the state = added then the entity state remains unchanged and nothing gets inserted into the database.

◦ **Le** says:

September 25, 2013 at UTC

You need to decorate the State property with the [NotMapped] attribute or you can specify this property to be ignored by configuration.

```
1 public class Customer
2 {
3     public int CustomerID { set; get; }
4     public string FirstName { set; get; }
5     public string LastName { set; get; }
6     [NotMapped]
7     public int Age { set; get; }
8 }
```

or

```
1 protected override void OnModelCreating(DbModelBuilder modelBuilder)
2 {
3     modelBuilder.Entity().Ignore(t => t.LastName);
4     base.OnModelCreating(modelBuilder);
5 }
```

35. **blueridge Crossfit** says:

September 25, 2013 at UTC

Hey There. I found your blog using msn. This is a really well written article. I'll be sure to bookmark it and return to read more of your useful info. Thanks for the post. I'll definitely comeback.

Reply

- **Le** says:
October 1, 2013 at UTC
Thanks for the positive feedback!

36. **Jose Gregorio Taveras** says:

September 28, 2013 at UTC

Hi, first let just say that this article it's a must have in the bookmark list of every .net developer great great article.

I have a question, what I should do if I want to join several entities inside a repo, i notice that using an extension methods on the Interface was the way but how can i access other entities ??

Reply

- **Le** says:
October 1, 2013 at UTC
If you really too, you can request for the IUnitOfWork via the Service Locator pattern in your Repository or Service class, e.g. <https://genericunitofworkandrepositories.codeplex.com/SourceControl/latest#dev/Northwind.Repository/CustomRepository.cs>. I've added a psuedo/fictitious example for you, you may want to download the project to see what are all the NuGet packages required.

37. **Jakob Jensen** says:

September 29, 2013 at UTC

Absolutely brilliant post.

I just got a error trying to run the program.

In the NorthwindContext it says it cant find DbContextBase and IDbContext.

That is also resulting in error in the constructor and the base.Set

Hope you can help

Jakob

Reply

- **Le** says:
September 29, 2013 at UTC
Is this the solution that was downloaded from: <https://genericunitofworkandrepositories.codeplex.com> ? Please make sure you are using this solution, and also do not forget to enable NuGet Package Restore at the solution level before running the project.

38. Pingback: [Reading Notes 2013-09-29 | Matricis](#)39. Pingback: [Reading Notes 2013-09-30 | Matricis](#)40. **frankfajardo** says:

September 30, 2013 at UTC

Hi Le,

Thanks for your detailed post. This is probably a silly newbie question: why do you have the EntityBase class in your Repository project? Wouldn't it make more sense to put it in Entity since all (or most of) your POCOs derive from it? Same goes with ObjectState and IObjectState

Reply

- **Le** says:
October 1, 2013 at UTC
Great question...! We deliberately placed EntityBase in the Repository layer/project to make the framework portable. Meaning I can easily add the Repository project into any solution and immediately be up and running with the UoW and Repo framework (Repo is completely decoupled from Entity). If we were to have EntityBase in the Northwind.Data project (which would now make Repo and Data coupled, which we don't want), which are really domain entities that are specific and coupled to an application, we would have to remember to add the EntityBase separately every time. Some of the design principles were re-usability, portability (plug-in play), easiability (plug-in play), modularity, and scalability (ability to scale this framework across any time of application (ASP.NET Web Forms, ASP.NET MVC, WPF, Silverlight, Windows Service, WCF, etc.).

41. **Vish** says:

October 9, 2013 at UTC

Hey Le,

I've been an ardent reader of all your posts. Brilliant stuff!

However, I've had 2 questions which have been niggling me for a while now and I haven't found a satisfactory response for them.

1] What's the advantage of using the EF, if I have a db. where all the stored procedures are already defined?

2] As far as the repository pattern goes, everything I have read/seen so far has 1 repository for 1 entity .i.e. a 1:1 relationship. In this case, what happens if the business layer needs results that span across entities – inner/left/outer join. In which repository would this API live?

Reply

- **Le** says:

October 10, 2013 at UTC

1. EF is an ORM does alot of the heavy lifting of mapping the data from SQL database into your objects/classes/entities, obviously this is an extremely over simplification on what EF's capabilities are, but the net answer is it makes the communication/interactivity between your app and your database much easier.

2. It should live in what ever repository that makes the most logical sense to you, this is very subjective and you become better at making decisions with experience. When you use EF, you can access all the other entities that are associated to it e.g. Customer.Orders.OrderDetails, etc. Now if you need access to entities that are not associated, then you need to create a sevice e.g. ICustomerService and inject IUnitOfWork into so that you can access whatever you need. Repositories are usually very focused on the entity at hand for separation of concerns, organization and manageability.

- **Vish** says:

October 10, 2013 at UTC

Thanks much!

12. Pingback: [*Upgrading to Async with Entity Framework, MVC, OData AsyncEntitySetController, Kendo UI, Glimpse & Generic Unit of Work Repository Framework v2.0 | Long Le's Blog*](#)

13. Pingback: [*MVC 4, Kendo UI, SPA with Layout, View, Router & MVVM – Part 1 | Long Le's Blog*](#)

14. Pingback: [*MVC 4, Web API, OData, Entity Framework, Kendo UI, Binding a Form to Datasource \(CRUD\) with MVVM – Part 3 | Long Le's Blog*](#)

15. Pingback: [*Lindermann's Blog | Excelente Material Sobre Implementação de MVC com Entity Framework*](#)

16. **frankfajardo** says:

October 13, 2013 at UTC

Hi Le, Me again. I could not find where you reset your entity's ObjectState after persisting the data changes. So I assume you don't. If that assumption is correct, was that intentional?

Reply

- **frankfajardo** says:

October 13, 2013 at UTC

And as a follow-up, why not just update EntityState directly? Thanks very much again for your help.

- **Le** says:

October 13, 2013 at UTC

Could you elaborate a bit more on "directly"? Because if it's dealing with a single entity on insert or update that's fine but we give up on entity graphs and that's not something we want. The optimal and easiest way without requiring the developer to understand and intimately know the different combinations and permutations of how EF deals entity graphs is to let the developer explicitly set this on each entity.

- **Le** says:

October 13, 2013 at UTC

Hmm, I'll probably right a blog post on this topic: "How EF deals with graphs".

- **Le** says:

October 13, 2013 at UTC

That's actually a good catch, hasn't been a problem for us yet, since we're always explicitly handling State, meaning when we're explicitly setting the entities State before every time before calling Save again. Please share your solution, we may incorporate it back into the framework!

- **frankfajardo** says:

October 13, 2013 at UTC

Hi Le,

Here's what I did:

- I'm not using the InsertGraph. And since the only time I need to tell EF the state of an entity is during an Update, I created a method on the DbContext whose sole purpose is to mark an entity as Modified.

- From my generic repository, I call that DbContext method from the Update(), right after Attach().

However, if I use the InsertGraph(), I think this solution will work instead:

- On the ApplyChanges, I simply add a line that changes the entities ObjectState property to Unchanged, after the ConvertState().

Thanks

Frank

o **Le** says:

October 13, 2013 at UTC

-Although we recommend that the State be explicitly set, we are automatically setting the root entity when performing an insert or update in the latest version of the framework ([v2.0](#)). Is this what you are doing in your mention of "method on the DbContext whose sole purpose is to mark an entity as Modified"?

```

1 public virtual void Update(TEntity entity)
2 {
3     dbSet.Attach(entity);
4     ((IObjectState)entity).State = ObjectState.Modified;
5 }
6
7 public virtual void Insert(TEntity entity)
8 {
9     dbSet.Attach(entity);
10    ((IObjectState)entity).State = ObjectState.Added;
11 }
```

-For the ApplyStateChanges, we will go ahead and add that as a WorkItem to reset the all the entities in the graph, after Save() has been called, thanks for the pointing this out.

o **frankfajardo** says:

October 13, 2013 at UTC

Le,

This is the method I added in the DbContextBase. It is actually an idea I took from Kazi Manzur Rashid's blog about Generic Repositories:

```

public void MarkAsModified(Tentity instance)
where Tentity : class
{
    Entry(instance).State = System.Data.EntityState.Modified;
}
```

17. **Brad** says:

October 16, 2013 at UTC

Good morning Le,

Thank you for an excellent article. It has very much helped me get a view of what I want to implement. I do have a couple of questions for you:

1. Is the Unit of Work (UoW) pattern worth the added complexity to the project (esp. with regards to having it somewhat already implemented through EF)? I feel like it is, but am having a hard time verbalizing why I feel that way, and wanted to get your thoughts.
2. It feels wrong to have the EF assemblies referenced in our IDbContext interface... Have you or any of your followers come up with a good way to extract that out to only be referenced in the Data assembly?

Thanks much!

Reply

o **Le** says:

October 16, 2013 at UTC

Hi Brad, thanks for the positive feedback.

1. The UoW pattern is suppose to help with shielding the developer from some of the complexities of knowing all the ins and outs EF e.g. managing entity graphs when inserting or updating them. It's also there so that you don't need EF referenced in our presentation layer e.g. ASP.NET MVC, we have it referenced only because Unity (DI & IoC) need so we can register the binding between NorthwindContext and IDbContext. The next version this will be completely removed and we will let each of their projects manage their own Unity Registrations, right now the Web App is handling all of this. Other than that there is not code in the presentation layer that depends on EF.

2. Our first objective was having no dependencies on EF in our presentation layer, perhaps we look at further pushing EF to only be referenced in the Data project, for the next version as well, thanks for the request.

18. **Quoc Nguyen** says:

October 17, 2013 at UTC

hi Le,

Thanks for great article but i'm stuck on step how to generate Model and Mapping.
Can you show me how to do that?

Thanks

Reply

◦ **Le** says:

October 18, 2013 at UTC

Hi Quoc, we used Entity Framework Power Tools from the Entity Framework Team.

[Blog at WordPress.com.](#) | [Customized Splendio Theme.](#)