Introduction

Easylink is a ORM framework that make time-consuming, tedious and error prone data access programming experience much simpler. With Easylink, programmers write less code and produce less bugs. Plus, the data access now is included in Easylink, programmer's time and energy can be better saved for UI and BL development. This adds more business values to clients.

Easylink is not Micro ORMs, such as FluentData or Dapper. Micro ORMs embed Sql query in NET code. Easylink builds query statement on the fly using object relation mapping configurations.

Compared to regular ORMS such as NHibernate and Entity framework, Easylink is easy to learn and use, fast and much smaller. In addition, Easylink makes added-values such as auditing, database test, validation, sql trace that are not found in those ORMs.

At the time of writing, Easylink supports Oralce, Sql Server, MySQL and PostgreSql.

Background

Applies to NET developers, architects, team leads who are looking for easy to use, fast, lightweight ORM.

How it works?

As an ORM component, Easylink serves as the data access layer between business logic and the database. BL never talks to the database directly but through Easylink.

1. To insert or update or delete a business object, BL calls Easylink, which uses Reflection to get property values from the business object. It then use mapping configurations to generate insert or update or delete statement, set Sql statement parameters accordingly, and execute commands against the database.

2. To retrieve business object(s), Easylink use mapping configuration to generate select statement, retrieve table and construct business object(s) and then pass to BL layer.

Why Easylink?

1. Easy to Use

Our philosophy behind Easylink is to make it as simple as possible. Easylink use fluent OR mapping to establish mapping between object and table. The following is Employee mapping class that map Employee class to table EMPLOYEE:

```
public class EmployeeMapping : Mapping
{
    public override IMappingConfig Setup()
    {
        var config = Class<Employee>().ToTable("EMPLOYEE");

        config.Property(e => e.Id).ToIdColumn("Id");
        config.Property(e => e.FirstName).ToColumn("FIRST_NAME");
        config.Property(e => e.LastName).ToColumn("LAST_NAME");
        config.Property(e => e.Active).ToColumn("ACTIVE");
        config.Property(e => e.LoginId).ToColumn("LOGIN_ID");
        config.Property(e => e.Role.Id).ToColumn("ROLE_ID");

        config.Link(e => e.Role).Join<Employee,Lookup>(e=>e.Role.Id, l => l.Id);

        return config;
    }
}
```

Following are Easylink CRUD API examples:

- database.Insert(employee)
- database.Update(employee)
- database.Delete(employee)
- var employee = database.RetrieveObject<Employee>(e=>e.Id== employeeId)

2. Support Object Link

When we retrieve a domain object via ORM, we want not only this object, but its child objects as well. It is not a hard thing to so, we can retrieve the parent object first, because parent object contains child object ID, we can retrieve each child object, and compose the whole domain object.

For instance, there are two domain classes: Employee and Address, and Employee has an Address, the pseudo code is as follows:

```
class Employee
{
    public long EmployeeId { get; set;}

    .....

   public Address Address  {get;set; }
}
```

To retrieve employee with address, we can use the following code:

```
var employee =    database.RetrieveObject<Employee>(e=>e.Id== employee.Id);

employee.Address =    database.RetrieveObject<Address>(a=>a.Id ==employee.Address.Id);
```

In this simple case, we call database twice to get the whole employee object. If the object contains X child objects, to get the object and its child objects, we need to make X+1 database calls. This cause severe performance hit when it comes to retrieving a large number of objects and one big reason that Micro-ORMs or ADO.NET are chosen over ORM.

With object link feature, Easylink solves this problem elegantly. User can just call

```
var employee =    database.RetrieveObject<Employee>(e=>e.Id ==employee.Id);
```

Easylink make one single database call to retrieve the whole employee object (Employee and Address).

To make this happen, all you need is to add a link to employee mapping configuration as follows:

```
...
  config.Link(e => e.Address).Join<Employee, Address>(e => e.Address.Id, a => a.Id);
...
```

This link joins Employee and Address together. Easylink can use this link information to create inner join when build select sql for Employee.
With object link feature, you end up with faster performance and less code.

3. Support Database Test

Automated tests are crucial in agile development process. Business always have high demand for delivering higher quality code in a much less time. Without automated tests, a tiny code change, would have to go through time consuming regression test to ensure that the code behaves as it used to. With automated tests, we can deliver fully tested code to production in much less time.

That said, to unit test business applications, which typically involves database operations, is not easy. To make each test independent and repeatable, each test should leave no "footprint" in the database whether that test is passed or not, or throw exception. To achieve this, the test code must be wrapped in a transaction, and the transaction must be roll back in the end no matter if the test succeeds or not.

Easylink supports database test with a method named ExecuteInTest. This method accepts test code as a function parameter. You can put as many transactional BL methods as you like in the test code.

No matter if test fails or succeeds or throws exception, the transaction rollback regardlessly, so all tests leaves no "footprint" in the database. You can create as many tests as you like, without worrying that one test could affect others. In addition, you can also port the test code to dev or test or even production database with no fears, as the tests are totally "harmless".

Following figure is a snapshot of the test for employee insert. No matter if insert fails, or retrieve fails or succeeds, the whole transaction rollback regardless. Test execution makes no footprint in the database, and therefore can be repeatable.

```
[TestMethod]
public void employee_should_able_to_insert()
{       Action proc = () =>
        {
            //insert employee
            var employee = FakeEmployee.GetFakeEmployee();
            database.Insert(employee);

            //retrieve
            var employeeInserted = database.RetrieveObject<Employee>
                                   (e=>e.Id== employee.Id);

            Assert.IsTrue(employeeInserted.LoginId == employee.LoginId);

        };

        database.ExecuteInTest(proc);
}
```

4. Support Validation

Validation is a must have feature for almost all business applications. Easylink makes validation extremely easy. It supports standard validation rules such as Required, Numeric, Maximum Length, Minimum Length, Fixed Length, Contains No Space, Email, Phone, Postal Code etc.

Each rule contains its own validation logic and can be reused across domain models. For instance, required rule can be applied to AccountNumber of BankAccount model, also it can be applied to FirstName of Employee model.

We setup validation rules in the constructor of domain model. Later, we validate object by checking if those rules are broken or not.

In the following example, a few validation rules are set up in the Employee constructor.

- FirstName is required, contain no space, and no more than 20 letters long.
- LastName should contain no space.
- Salary must be more than $10,000

```
public Employee()
{

    AddRules<Employee>(e => e.FirstName, "First Name", new RequiredRule(),
                                        new MaximumLengthRule(20),
                                        new ContainsNoSpaceRule());

    AddRules<Employee>(e => e.LastName, new ContainsNoSpaceRule());

    AddRules<Employee>(e => e.Salary, new ThresholdRule(">", 10000));


}
```

Apart from standard rules, Easylink also support custom rule. Users can write their own validation logic. Easylink use both custom and standard validation rules together to validate against the object.

5. Support Auditing

Auditing is required for some business scenarios due to security or compliance.

Easylink provides robust audit support. With Easylink, you do not need to write your own auditing logic any more. While executing CRUD operations, Easylink saves audit data such as operation, operation date, and user who performing the operation to audit table.

Below is the Easylink example of auditing employee record 4813 insert, update and delete.

Results | Messages

	USER_ID	TABLE_NAME	RECORD_ID	DESCRIPTION	OPERATION	TIMESTAMP
1	dzhou	EMPLOYEE	4813	Inserted FIRST_NAME=david, LAST_NAME=smith, SALARY=12045.27, EMPLOYED_SINCE=...	Insert	2014-11-28 15:45:44.330
2	dzhou	EMPLOYEE	4813	Updated FIRST_NAME from [david] to [New Name], DEPARTMENT_ID from [2] to [4].	Update	2014-11-28 15:45:50.670
3	dzhou	EMPLOYEE	4813	Deleted FIRST_NAME=New Name, LAST_NAME=smith, SALARY=12045.27, EMPLOYED_SI...	Delete	2014-11-28 15:45:51.780

Because auditing is not a feature that comes without cost, Easylink allows developers to only audit those domain models that they want to. For instance, if you want to audit Employee class, just mark the class with [Audit] attribute.

If you want to enable audit feature at application level, you need to tell Easylink what AuditRecord type to use when initializing Easylink.

```
DatabaseFactory.Initialize<AuditRecord>(@"your connection string",
                                        DatabaseType.SqlServer, "dbo");
```

If you do not need Audit feature at application level, you can simply call

```
DatabaseFactory.Initialize("your connection string", DatabaseType.SqlServer, "dbo");
```

6. Support Major Databases

So far, Easylink supports Oralce, SqlServer, MySql, and PostgreSql.

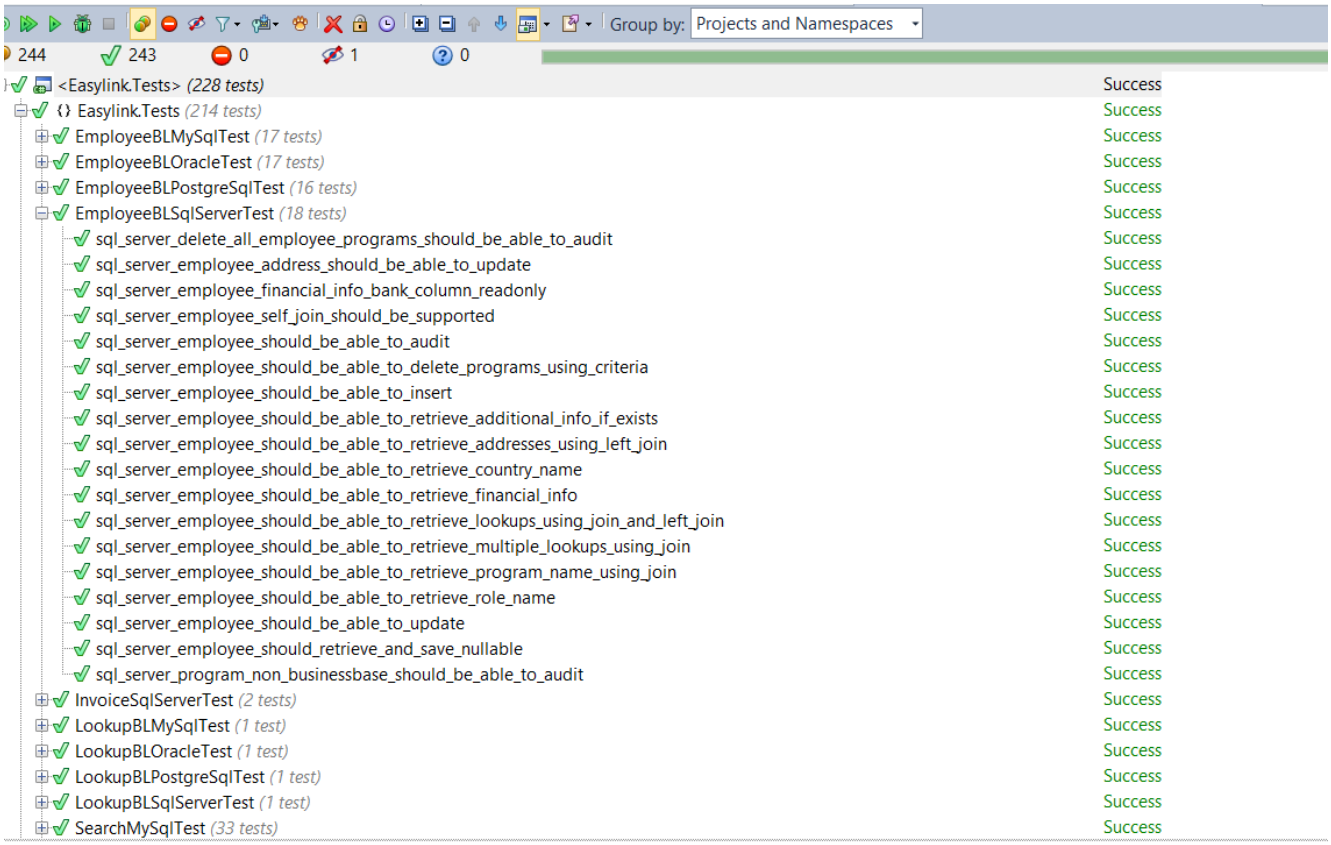
Since Easylink provides an abstraction between your code and database. There is no need to learn APIs of a specific database provider (like ODP.NET). You use the same Easylink interface to access all databases that Easylink supports.

This makes Easylink a good choice for data migration between different databases.

7. Code Quality Guaranteed

Since Easylink is a framework product, we understand the quality of the product really matters.

Easylink has built-in unit test code to ensure CRUD operations, query search, validation, auditing are correct against Oracle, SqlServer, MySql and PostgreSql.



Feedbacks

There has been a long debate in the developer community, whether to use micro ORM or heavy ORMs such as EF or NHibernate. Using Micro ORM properly, you code run faster, and you are more flexible in tuning your Sql. However, you could end up writing a lot more code, which could potentially introduce more bugs.

On the other hand, to use EF, or NHibernate, in addition to performance overhead, you have to bear with the fact that 80% features you never know and never care, yet you have no choice but dance with their pace and upgrade your applications (sometimes

unnecessarily) when new version are coming. Learning curve is also another reason that some programmers choose to stay away from those ORMs.

Easylink is a viable option, it is small, faster, easy to use, powerful, and support major databases. You can easily install Easylink in your NET solution.
In Package Manager Console of your Visual Studio, simply typing
"Install-Package Easylink"

Your feedbacks are important to us. Whether you agree with us or not, we want to hear from you! It you want to, please call us at 403-397-3186 or send us email at daniel.zhou@easyexpresssoft.com.