|  |
| --- |
| Megadotnet |
| Iron Framework Developer’s Guide |
| Draft for preview |

|  |
| --- |
| Peter Liu  2011-08-28 |

Table of content

[Introduction 3](#_Toc307844695)

[Technology and Software Requirements 3](#_Toc307844696)

[Design Goals and Non-Goals 4](#_Toc307844697)

[Goals 4](#_Toc307844698)

[Non-Goals 4](#_Toc307844699)

[Architecture 4](#_Toc307844700)

[Design Principles 4](#_Toc307844701)

[Architectural diagram 5](#_Toc307844702)

[Data access infrastructure 7](#_Toc307844703)

[Repository Class Diagram 7](#_Toc307844704)

[Business layer 7](#_Toc307844705)

[Service layer 8](#_Toc307844706)

[RESTful Service 8](#_Toc307844707)

[Presentation layer 12](#_Toc307844708)

[Asp.net MVC Application 13](#_Toc307844709)

[Asp.net Web Form Application 13](#_Toc307844710)

[Cross cutting 13](#_Toc307844711)

[Common and Utility 13](#_Toc307844712)

[Unit Test 13](#_Toc307844713)

[For Asp.net MVC application 13](#_Toc307844714)

[For components 13](#_Toc307844715)

[Code generation 14](#_Toc307844716)

[T4 (Text Template Transformation Toolkit) template 14](#_Toc307844717)

[Deployment 14](#_Toc307844718)

[System Quality Attributes 14](#_Toc307844719)

[Reusability 14](#_Toc307844720)

[Testability 15](#_Toc307844721)

[Scalability 15](#_Toc307844722)

[Maintainability 15](#_Toc307844723)

[Glossary/Terms 15](#_Toc307844724)

# Introduction

Iron Framework focus on enterprise solution based on Microsoft .net framework 4 RTM version. Combine with Entity Framework, Enterprise Library, WCF, Asp.net MVC   
components implement:

* + Layered Architecture
  + DDD (Domain Driven Design),
  + AOP (Aspect-oriented programming)
  + Service-Oriented Architecture

There architectural styles offer flexible extension and rapid developed infrastructure.

# Technology and Software Requirements

Technology

1. C#
2. WCF
3. ASP.Net
4. Silverlight
5. Visual Studio 2012/2013
6. TFS 2010/2013 (option)
7. Web Deployment Server/ IIS 7.5
8. Windows Server 2008 R2/2012 (option)
9. SQL Server 2008/2012 Enterprise(option)

Components and libraries:

1. ADO.Net Entity framework
2. Enterprise Library
3. Newtonsoft.Json for .net
4. Moq
5. ASP.NET MVC RTM
6. WCF Web API Preview
7. ASP.NET WebAPI
8. The SQL Server AdventureWorks sample database (option)

# Design Goals and Non-Goals

## Goals

Using emerging technique and extremely popular architectural style based on Microsoft .net platform to construct enterprise common rapid developed framework. Demonstrate some reuse library and coding trick and design skill that we have accumulated.

## Non-Goals

It is not cover all about Domain Driven Design (DDD). Domain Driven Design is much more than Architecture and Design Patterns. It implies a specific way of working for development teams and their relationship with Domain experts, a good identification of Domain Model elements (Aggregates/Entity Model, etc.) based on the Ubiquitous Language for every Model we can have, identification of Bounded-Contexts related to models, and a long etcetera related to the application life cycle that we are not covering.

# Architecture

## Design Principles

**De-coupling between Components**

The de-coupling techniques are based on the Principle of Dependency Inversion, which sets forth a special manner of de-coupling, where the traditional dependency relationship used in object orientation (high-level layers should depend on lower-level layers), is inverted. The goal is to have layers independent from the implementation and specific details of other layers, and therefore, independent from the underlying technologies as well.

There are several techniques and patterns used for this purpose, such as Plug-in, Service Locator, Dependency Injection and IoC (Inversion of Control).

Basically, the main techniques we propose to enable de-coupling between components are:

- Inversion of control (IoC)

- Dependency injection (DI)

**Architectural Patterns and Design Patterns in project**

* Model View Controller (MVC)
* Model View View Model (MVVM)
* Repository
* Unit of Work/Context (UoW)
* Lazy Load
* Eager Load
* Domain model
* Server layer
* Data Mapped
* Data Transfer Object (DTO)
* Plain old CLR object (POCO)

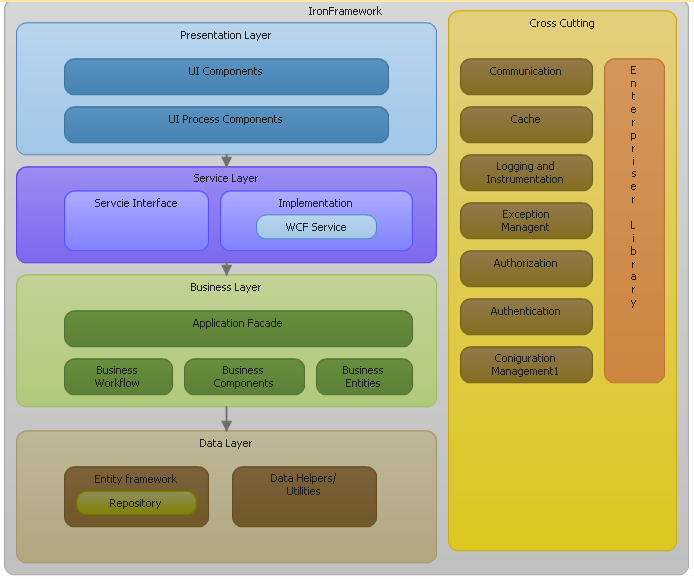
**Basic Principle**

* SOLID
* DRP
* KISS
* The separation of Concern

You have to know above design patterns and architectural patterns that help you understand architect of framework.

## Architectural diagram

Below is a cross-section that shows some of the technologies used and the communication between the layers. The right-hand-side shows some of the key design patterns used in the reference application.





The four tiers or layers are:

* Presentation layer — handles external interaction with the user
* Service layer — interactive with presentation layer and provider a data service
* Business layer — manipulates the information required by the user
* Database layer — stores the data handled by the system

We have two leveled IOC/DI they are data access layer and business layer in project. Each layer use own DI container. Business object interface register object with xml file to unity container. Data access interface register object in memory to unity container. You may unify one container or nested container to resolve all objects.

For business object we prefer use AOP approach that block executed method here then to do common function. It is usually called crossing concern. We have sample demo in unit test project.

If you do not need the Service layer, UI layer can be directly invoke Business layer that performance would be more efficiently.

Enterprise library 5 has remove caching call handler base on Http Runtime Cache. We have implemented it by .net framework 4 System.Runtime.Caching.

General, exception logging will locate as ErrorRolling.logfilename filename format at BO Service folder. It is at bin folder for Unit test project. It is also specific custom output path.

**Technical features:**

* Asp.net MVC Razor engine
* Dependency injection with Asp.net MVC
* Asp.net WebForm support the strongly typed data binding model and custom paging data server controls.
* Hierarchical data model work with JQuery Plug-in.
* Microsoft .Net Framework 4 new Caching engine support
* Entity framework POCO and T4 template code generation
* Cross cutting with Enterprise library application block
* Dependency injection with Unity DI container
* WCF RESTful service based on WCF 4 Service
* Repository and Unit of work Pattern.
* Unit test with the Moq framework

## Data access infrastructure

### Introduction

The Data Layer (DL) handles the persistence of Business Objects. It offers two providers: ADO.NET and ADO.Net Entities Framework. Today the Entity Framework source code is being released under an open source license (Apache 2.0), and the code repository will be hosted on CodePlex to further increase development transparency.

*IRepository.tt* template will generate those data access class. By default, data access type provider work with Entity Framework. *AdventureWorks.Context.tt* template will generate entity framework data access context type. We have modified the default the ADO.NET DbContext Generator template which from Entity framework 5 then added WCF data member attribute. It is can get it from NuGet.   
 In the Web.config/App.Config file you may indicate which one to use. ADO.NET implements a 'data provider factory' which uses an abstract factory pattern and returns database specific singleton factories. You may implement your own data access layer with data repository interface.

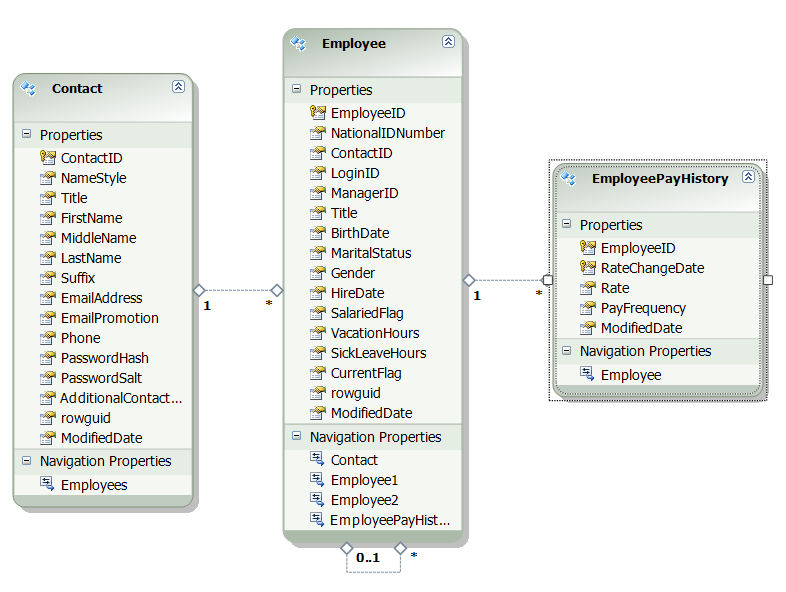
<connectionStrings>

<add name="AdventureWorksEntities" connectionString="metadata=res://\*/AdventureWorks.csdl|res://\*/AdventureWorks.ssdl|res://\*/AdventureWorks.msl;provider=System.Data.SqlClient;provider connection string=&quot;Data Source=.;Initial Catalog=AdventureWorks;Integrated Security=True;MultipleActiveResultSets=True&quot;" providerName="System.Data.EntityClient" />

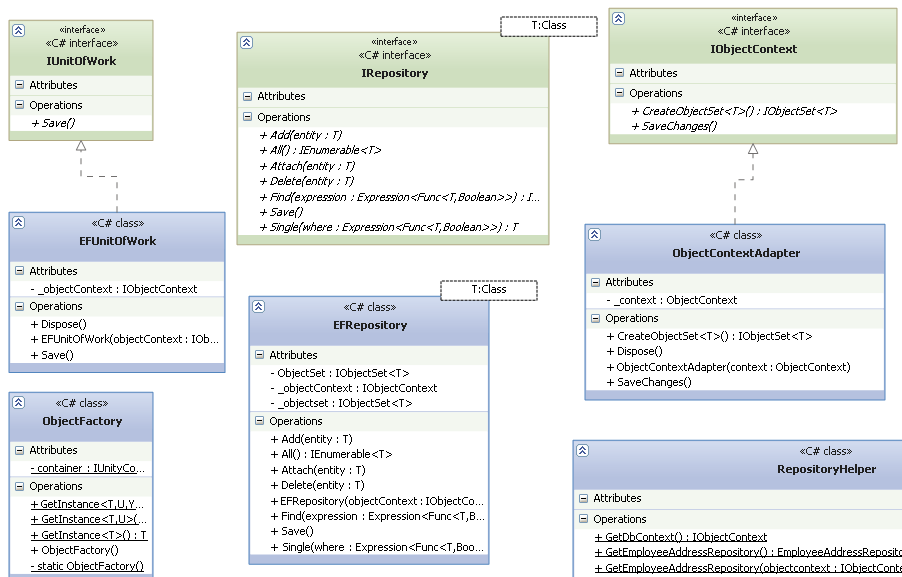
</connectionStrings>

Note: Those templates depend on EDMX file that based on your database model design first.

Database model example (snapshot from AdventureWorks.edmx):



### Repository Class Diagram

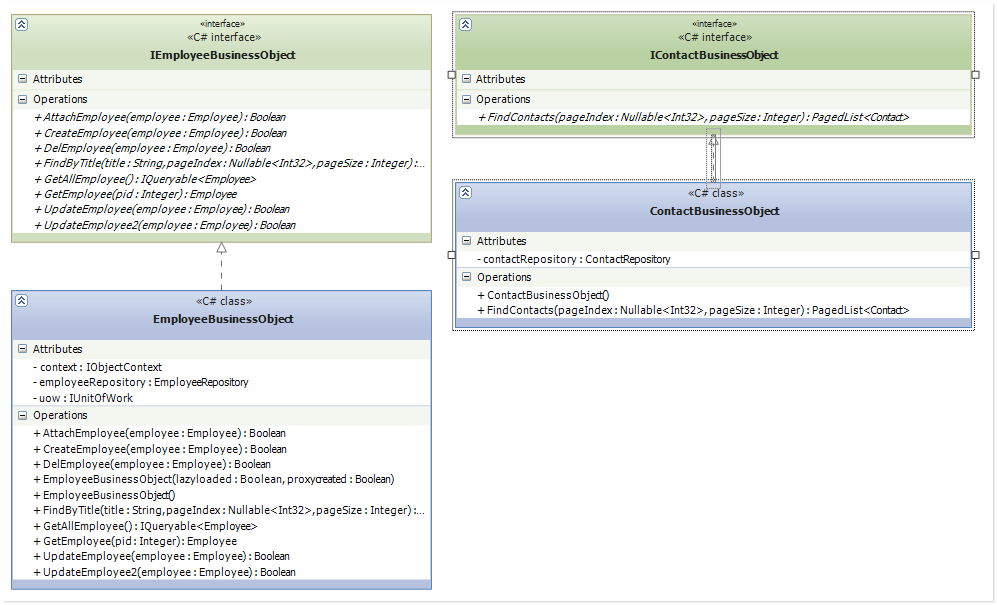


## Data transfer Object layer

Why DTOs? DTOs are simple objects that should not contain any business logic that would require testing. DTOs are most commonly used by the Services layer in an N-Tier application to transfer data between itself and the UI layer. The main benefit here is that it reduces the amount of data that needs to be sent across the wire in distributed applications. They also make great models in the MVC pattern. The framework support transfer the Entity framework model to View Model. DTO Object used to the View Model for presentation. The difference between data transfer objects and business objects or data access objects is that a DTO does not have any behavior except for storage and retrieval of its own data (accessors and mutators).

## Business Object layer

Using *AdventureWorks.tt* T4 template generate entities POCO mode class as business entities. Business object project include business rules and business objects. We put the core business logic in this project.

Business Objects encapsulate business logic in the form of business rules. Business Objects themselves have no knowledge about databases or data persistence, which is handled by the Data access layer. 

## Service layer

It is implemented by WCF 4 service. It is include data contract interface and service implement. The WCF Service Layer receives messages from the PL. It interprets the message, unpacks the Data Transfer Objects, and orchestrates and coordinates the interaction between Business Objects and Data Access Objects.

We will expose the OData API structure in the future.

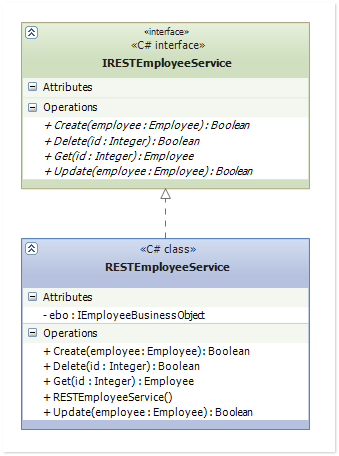
### RESTful Service

It is implemented REST Service by WCF 4 Web API and Asp.net Web API

WCF Web API history

For several years now the WCF team has been working on adding support for REST. This resulted in several flavors of REST support in WCF: WCF Web HTTP, WCF REST Starter Kit, and then finally WCF Web API. In parallel the ASP.NET MVC team shipped support for building basic web APIs by returning JSON data from a controller. Having multiple ways to do REST at Microsoft was confusing and forced our customers to choose between two partial solutions. So, several months ago the WCF and ASP.NET teams were merged together and tasked with creating a single integrated web API framework. The result is ASP.NET Web API.

Here is a simply CRUD method in it. The Class diagram like this:



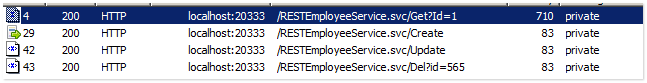
We use Fiddler test it, assume you have deployed Service from URL:

<http://localhost:20333/RESTEmployeeService.svc>

When debug REST service we suggest you set configuration section like this:

<serviceDebugincludeExceptionDetailInFaults="true" />

following snapshot for test CRUD method from fiddler:



#### Http GET

We send HTTP GET request like this:

GET http://localhost:20333/RESTEmployeeService.svc/Get?Id=1 HTTP/1.1

User-Agent: Fiddler

Content-Type: application/xml; charset=utf-8

Host: localhost:20333

Content-Length: 706

It responses like this RAW text:

HTTP/1.1 200 OK

Server: ASP.NET Development Server/10.0.0.0

Date: Mon, 27 Jun 2011 03:01:25 GMT

X-AspNet-Version: 4.0.30319

Content-Length: 710

Cache-Control: private

Content-Type: application/xml; charset=utf-8

Connection: Close

<Employee xmlns="http://schemas.datacontract.org/2004/07/BusinessEntiies" xmlns:i="http://www.w3.org/2001/XMLSchema-instance"><BirthDate>1972-05-15T00:00:00</BirthDate><ContactID>1209</ContactID><CurrentFlag>true</CurrentFlag><EmployeeID>1</EmployeeID><Gender>M</Gender><HireDate>1996-07-31T00:00:00</HireDate><LoginID>adventure-works\guy1</LoginID><ManagerID>16</ManagerID><MaritalStatus>M</MaritalStatus><ModifiedDate>2004-07-31T00:00:00</ModifiedDate><NationalIDNumber>14417807</NationalIDNumber><SalariedFlag>false</SalariedFlag><SickLeaveHours>30</SickLeaveHours><Title>Production Technician - WC60</Title><VacationHours>21</VacationHours><rowguid>aae1d04a-c237-4974-b4d5-935247737718</rowguid></Employee>

#### Http POST

It is usually use for Create specific object. We send request RAW text like this:

POST http://localhost:20333/RESTEmployeeService.svc/Create HTTP/1.1

User-Agent: Fiddler

Content-Type: application/xml; charset=utf-8

Host: localhost:20333

Content-Length: 618

<Employee xmlns="http://schemas.datacontract.org/2004/07/BusinessEntiies" xmlns:i="http://www.w3.org/2001/XMLSchema-instance"><BirthDate>1972-05-15</BirthDate><ContactID>1209</ContactID><CurrentFlag>true</CurrentFlag><Gender>M</Gender><HireDate>1996-07-31T00:00:00</HireDate><LoginID>adventure-works \guy12</LoginID><ManagerID>16</ManagerID><MaritalStatus>M</MaritalStatus><ModifiedDate>2004-07-31</ModifiedDate><NationalIDNumber>14409807</NationalIDNumber><SalariedFlag>false</SalariedFlag><SickLeaveHours>30</SickLeaveHours><Title>Production Technician - WC60</Title><VacationHours>21</VacationHours></Employee>

It response RAW text like this:

HTTP/1.1 200 OK

Server: ASP.NET Development Server/10.0.0.0

Date: Mon, 27 Jun 2011 03:13:57 GMT

X-AspNet-Version: 4.0.30319

Content-Length: 83

Cache-Control: private

Content-Type: application/xml; charset=utf-8

Connection: Close

<boolean xmlns="http://schemas.microsoft.com/2003/10/Serialization/">true</boolean>

#### Http PUT

Request RAW:

PUT http://localhost:20333/RESTEmployeeService.svc/Update HTTP/1.1

User-Agent: Fiddler

Content-Type: application/xml; charset=utf-8

Host: localhost:20333

Content-Length: 621

<Employee xmlns="http://schemas.datacontract.org/2004/07/BusinessEntiies" xmlns:i="http://www.w3.org/2001/XMLSchema-instance"><BirthDate>1972-05-15</BirthDate><ContactID>1209</ContactID><CurrentFlag>true</CurrentFlag><EmployeeID>565</EmployeeID><Gender>M</Gender><HireDate>1996-07-31T00:00:00</HireDate><LoginID>adventure-works \guy12</LoginID><ManagerID>16</ManagerID><MaritalStatus>M</MaritalStatus><ModifiedDate>2010-06-01</ModifiedDate><NationalIDNumber>14409807</NationalIDNumber><SalariedFlag>false</SalariedFlag><SickLeaveHours>30</SickLeaveHours><Title>Slayer</Title><VacationHours>21</VacationHours></Employee>

Response RAW:

HTTP/1.1 200 OK

Server: ASP.NET Development Server/10.0.0.0

Date: Mon, 27 Jun 2011 03:20:13 GMT

X-AspNet-Version: 4.0.30319

Content-Length: 83

Cache-Control: private

Content-Type: application/xml; charset=utf-8

Connection: Close

<boolean xmlns="http://schemas.microsoft.com/2003/10/Serialization/">true</boolean>

#### Http DELETE

Request RAW:

DELETE http://localhost:20333/RESTEmployeeService.svc/Del?id=565 HTTP/1.1

User-Agent: Fiddler

Content-Type: application/xml; charset=utf-8

Host: localhost:20333

Content-Length: 621

Response RAW:

HTTP/1.1 200 OK

Server: ASP.NET Development Server/10.0.0.0

Date: Mon, 27 Jun 2011 03:21:19 GMT

X-AspNet-Version: 4.0.30319

Content-Length: 83

Cache-Control: private

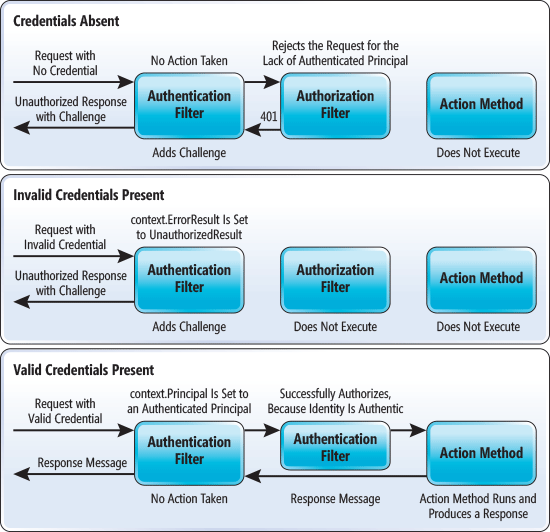
Content-Type: application/xml; charset=utf-8

Connection: Close

<boolean xmlns="http://schemas.microsoft.com/2003/10/Serialization/">true</boolean>

We also can use WCF Web API to do unit test with REST service.

### Web API



#### Authentication

Web API assumes that authentication happens in the host. For web-hosting, the host is IIS, which uses HTTP modules for authentication. You can configure your project to use any of the authentication modules built in to IIS or ASP.NET, or write your own HTTP module to perform custom authentication.

##### HMAC based authentication

HMAC (hash-based message authentication code) provides a relatively simple way to authenticate HTTP messages using a secret that is known to both client and server. Unlike [basic authentication](http://www.piotrwalat.net/basic-http-authentication-in-asp-net-web-api-using-message-handlers/) it does not require transport level encryption (HTTPS), which makes its an appealing choice in certain scenarios. Moreover, it guarantees message integrity (prevents malicious third parties from modifying contents of the message).

On the other hand proper HMAC authentication implementation requires slightly more work than basic HTTP authentication and not all client platforms support it out of the box (most of them support cryptographic algorithms required to implement it though).

## Presentation layer

The concern of the Presentation Layer (PL) is to present information in a consistent and easy-to-understand manner to the end-user. It includes fully functional implementations of three UI platforms: ASP.NET Web Form 4, ASP.NET MVC 3/4 and WPF (Windows Presentation Foundation). Each of UI platforms consumes the exact same services hosted under WCF. This service-oriented model is an implementation of the Application Facade Design Pattern. Applications designed this way have the ability to expose their Services (Web or otherwise) with no extra work (other than configuring the WCF host). WCF is truly a powerful new platform!

You will find asp.net MVC unit testing project in the solution. We may use the TDD approach develop it through whole life cycle.

### Asp.net MVC Application

Razor provides a great new view-engine option that is streamlined for code-focused tempting. It is syntax is compact and reduces typing – while at the same time improving the overall readability of your markup and code.

### Asp.net Web Form Application

One of features that implement the strongly typed data binding model and the custom paging data server controls.

## Cross cutting

It is implemented by Enterprise library and Unity. We use unity implement interception and Dependency injection then block method to do something that we wanted.

Unity configuration: constructor Parameters with EF DbContext Injection and lifecycle time

<register type="IEmployeeBusinessObject" mapTo="EmployeeBusinessObject" name="NoOLazyloadedAndProxyCreated">

<!--<lifetime type="singleton" />-->

<constructor>

<param name="lazyloaded" value="false"/>

<param name="proxycreated" value="false"/>

</constructor>

<interceptor type="TransparentProxyInterceptor" />

<policyInjection />

</register >

Case1: Cross cutting with Validation, Logging, Caching

[ValidationCallHandler]

[LogCallHandler(BeforeMessage = "before GetEmployee", AfterMessage = "after GetEmployee",

IncludeCallStack = true)]

//[CachingCallHandler(0, 10, 0, Order = 3)]

Employee GetEmployee(

[RangeValidator(1, RangeBoundaryType.Inclusive, int.MaxValue, RangeBoundaryType.Inclusive)] int pid);

## Common and Utility

It is include the DTO class of UI and service factory which is warp unity implement DI, AOP feature.

## Unit Test

We use the MS Test in the Unit test project. So you may also use Test Driven plug-in or the Jetbrains ReSharpr plug-in working on it conveniently.

Because application block work with plug mode of framework, please verify assemblies have reside against bin folder that test case of unit test work well.

### For Asp.net MVC application

It is implementing separated with Model unit test for Controller of the asp.net MVC application. In the future, we will implement unit test with View Model.

### For other components

It is including repository and WCF service unit test.

### Business Object with Data access object

In DALConfig.xml files, we may switch FakeContextAdapter object, like following xml section:

<register type="DataAccessObject.IObjectContext, DataAccessObject"

mapTo="DataAccessObject.FakeContextAdapter, DataAccessObject">

<interceptor type="InterfaceInterceptor"/>

</register >

Also keep App Setting key ‘UsingXmlConfigForUnity’ as true value

<appSettings>

<add key="UsingXmlConfigForUnity" value="true" />

</appSettings>

Finally, we may get results:

[Theory, AutoData, AutoRollback]

public void TestAdd(AddressDto \_AddressDto)

{

var \_AddressBO = new AddressBO(new FakeAddressConverter());

bool hasAdded=\_AddressBO.CreateEntiy(\_AddressDto);

Assert.True(hasAdded);

}

1 passed, 0 failed, 0 skipped, took 1.83 seconds (xUnit.net 1.9.2 build 1705).

## Code generation

### T4 (Text Template Transformation Toolkit) template

In Visual Studio, a T4 text template is a mixture of text blocks and control logic that can generate a text file. The control logic is written as fragments of program code in Visual C# or Visual Basic. The generated file can be text of any kind, such as a Web page, or a resource file, or program source code in any language.

You will find other t4 template in other project. It is by default generate general simply logic code. You can modify it depends on your requirement.

# Deployment

First you need deploy Business object service as host in IIS or single process.

Then deploy website under IIS application folder.

**Building**

**Command-line**

Run this batch file from the directory where IronFramework.sln resides.

Usage: cmdbuild [debug | release ] [output\_path]

Example: cmdbuild debug c:\Ironframework-debug-build

# System Quality Attributes

## Reusability

It is intended to design The WCF service layer in a way that it is reusable by other applications as well.

## Testability

The amount of effort required to create these tests is directly related to the testability of the underlying software.

## Scalability

The system should scale to increased numbers of threads to be handled per unit time as well as when new forums are added.

## Maintainability

Maintainability is important and the systems would lend itself to easy maintenance including feature additions and bug fixes due to the clear separation of the components into layers.

# Appendix

## Source Code

[Asp.net MVC RTM framework](https://aspnet.codeplex.com/releases/view/58781)

[Asp.net Web](http://aspnetwebstack.codeplex.com/)

[Entity Framework](http://entityframework.codeplex.com/)

[Enterprise Library](http://entlib.codeplex.com/SourceControl/list/changesets)

## Glossary/Terms

| Term | Definition |
| --- | --- |
| AOP | Aspect-oriented programming |
| DDD | Domain Driven Design |
| POCO | Plain old CLR object |
| T4 | Text Template Transformation Toolkit |
|  |  |
|  |  |
|  |  |
|  |  |