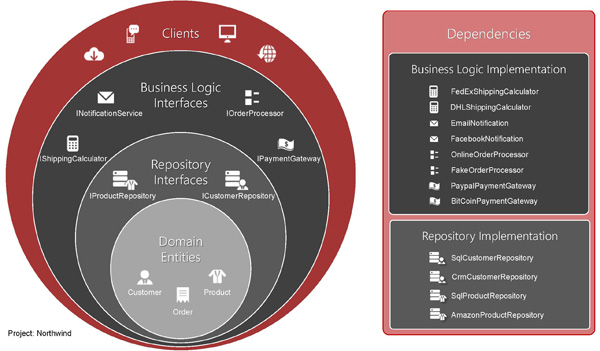
## SSW Data Onion

Integrate Entity Framework into your Onion Architecture Enterprise Application

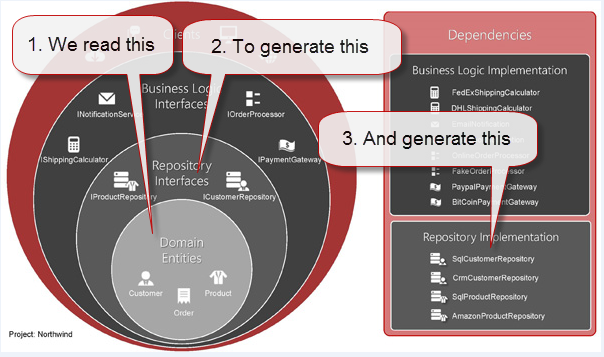


**Figure: The Layers of the onion architecture – as per ssw rule** [**http://rules.ssw.com.au/SoftwareDevelopment/RulesToBetterMVC/Pages/The-layers-of-the-onion-architecture.aspx**](http://rules.ssw.com.au/SoftwareDevelopment/RulesToBetterMVC/Pages/The-layers-of-the-onion-architecture.aspx)

At SSW, when building enterprise .net applications we follow the Onion Architecture as outlined above. This domain model centric approach encourages the development of maintainable solutions by integrating loosely coupled components via interfaces.

Adopting Onion architecture can lead to writing lots of extra code. SSW Data Onion makes this easy by generating an Entity Framework Code First implementation for persisting your Domain Entities.

In a nutshell:



1. SSW Data Onion contains a 3 .tt templates that reads domain classes from your appropriate domain assembly
2. We generate a repository interface for each domain class.
3. We generate a default EF Code first repository implementation for each domain class.
4. The DbContext also needs to reference every class so we also generate that.
5. We also have a set of core classes to support:
   1. Dependency injection of DBInitializers <http://www.entityframeworktutorial.net/code-first/database-initialization-strategy-in-code-first.aspx>
   2. Managing DbContext lifecycle in a unit of work pattern

## The packages

One core consideration for implementing this under the onion architecture was the ability to place all components in separate projects / assemblies. To allow this, SSW Data Onion is split across multiple Nuget packages. You can install these packages to separate projects (recommended) or you can install to one test project.

### SSW.Data.EF

This is SSW’s core Entity Framework package.

### SSW.Data.Entities

Provides some common Entity interfaces.

### SSW.Data.DbContext.Generator

Builds upon SSW.Data.EF to provide tt templates that generate a DbContext class

### SSW.Data.RepositoryInterfaces.Generator

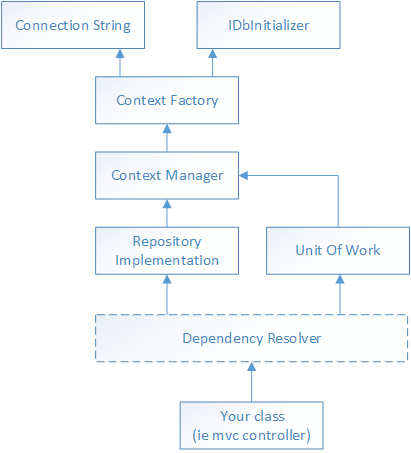
Builds upon SSW.Data.EF to provide tt templates that generate repository interfaces

### SSW.Data.Repositories.Generator

Builds upon SSW.Data.EF to provide tt templates that generate repository implementations.

## Dependency Injection Configuration

SSW Data Onion is intended for use with an IOC / Dependency Injection container. Each class receives its dependencies via constructor parameters. A typical dependency chain for a repository and a unit of work are diagrammed below. Behavior at each level can be changed by providing your own implementation of the corresponding interface.



**Figure: the dependency relationships between SSW Data Onion core classes.**

### IDbInitializer

An implementation of the standard System.Data.Entity.IDatabaseInitializer<T> interface from Entity Framework. Typically this is where we configure Entity Framework code-first migrations.

### Context Factory

An Implementation of the standard System.Data.Entity.Infrastructure.IDbContextFactory<T> interface.

The role of the ContectFactory in this design is to create new instances of your DbContext. We provide a default, generic implementation that makes the DbInitializer an explicit dependency.

<http://msdn.microsoft.com/en-us/library/hh506876(v=vs.113).aspx>

### Context Manager

Where Context Factory creates new DbContext instances, Context manager builds on this to manage the dbContext lifecycle. We provide a ContextManager implementation that is disposable: when the context manager is disposed, the underlying DbContext is disposed.

So we can control the lifecycle of our DbContexts by configuring the lifecycle of the Context Manager.

### Repository Implementation.

The repository implementations as generated by SSW.Data.Repositories.Generator simply depend upon the context manager.

### Unit of work pattern

The unit of work is designed to combine actions across multiple repositories into a single unit of work.

The implementation provided by SSW Data Onion supports multiple databases / DbContexts by depending upon a collection of context managers.

### Your Class

When it comes to using Data Onion, your code just needs a dependency on one or more repositories and a unit of work.

Advanced Features

Context-per-request pattern

Multiple databases with a unit of work.

Transaction Scope

Dropping DbContext for performance

Batch Inserts