**Accessing the database via KLAS code**

In our MVC Controllers we are leveraging “Dependency Injection” to handle our accessing of the database. Specifically we are using “[Ninject](http://www.ninject.org/) Dependency Injection framework” an IOC container. To learn more about Dependency Injection see <http://bobcravens.com/2010/03/dependency-injection-and-inversion-of-control-containers/> and <http://blog.agilistic.nl/a-step-by-step-guide-to-using-ninject-for-dependancy-injection-in-c-sharp/> . Also see <http://www.ninject.org/>

IOC containers add abstraction around the creation / life-cycle management process. This abstraction requires a certain level of understanding for a developer working on the code. Developers are much more used to using ‘new’ to create objects than wiring up the IOC container. Once this barrier to entry has been overcome, then clarity is not really an issue.

In our MVC Controllers you will notice the “[Inject]” attribute, followed by a property:

[Inject]

public EmployeeController EmployeeController { get; set; } – we don’t seem to ever get or set this property within the MVC Controller, so perhaps declaring it as a property is required by Ninject????

“EmployeeController” class belongs to our Domain.Controllers namespace, and it contains a property of a repository Interface type.

IEmployeeRepository \_employeeRepository (Declaring the prop as an Interface type seems to be related to Dependency Injection). In comparison the C:\Code\KLAS\Klas.Domain\Controllers\EcosystemController.cs declares \_ecosystemRepository as readOnly without the get; set; (so it’s a private variable and not a property). This variable (or property, as the case may be) is set in the constructor.

The “IEmployeeRepository” interface is implemented by the “EmployeeRepository” class which belongs to the Domain.Repositories namespace.

The Domain.Controller (i.e EmployeeController class) has methods that access the database, through the “IEmployeeRepository” variable (property)

public void AddEmployee(string firstName, string lastName, string userName)

{

\_employeeRepository.AddEmployee(firstName, lastName, userName);

}

\_employeeRepository.AddEmployee first defined (without any functionality or logic) in public interface IEmployeeRepository. The public class EmployeeRepository implements the interface and defines the functionality (logic) of AddEmployee.

class EmployeeRepository declares private KlasEntities \_dbContext; KlasEntities is an auto-generated class (file) from the EDMX. \_dbContext is set in the Constructor of EmployeeRepository

Within the Klas.Domain project there is a Ninject folder with a file (class) named KlasDependencies which creates a concrete implementation of the ‘NinjectModule’ abstract class. In there are many Bind statements for example: Bind<IEmployeeRepository>().To<EmployeeRepository>().InRequestScope();

The generic Ninject ‘Bind’ method takes the type of the service (‘IEmployeeRepository for instance) to bind. The Ninject ‘To’ method takes the implementation type (‘EmployeeRepository’). This wires up the Ninject container to provide an instance of ‘EmployeeRepository’ for every ‘IEmployeeRepository request. The reason for using InRequestScope () is to make sure that a single instance of an object is shared by all objects created via the Ninject kernel for that HTTP request (e.g. to share an object that is expensive to create).

**And as if by magic The declaration in the MVC controller gives access to the database!!!**