**Starting NHibernate (A beginners guide).**

**Introduction:**

The purpose of this article is to provide a basic introduction and guideline to start working with NHibernate. We will discuss the required technique and concepts first and later part of this article we will create a practical projects.

NHibernate is an ORM (Object-Relational mapper) for .net platform just like Entity Framework. But it is an open source tool supported by NHibernate community. Official links to NHibernate site is <http://nhibernate.info/>

If you do not know what ORM is please read following links:

<https://en.wikipedia.org/wiki/Object-relational_mapping>

<http://stackoverflow.com/questions/959270/what-is-nhibernate>

**How to use**

To use NHebernate into your application you have to perform following three steps

1. Install and add references of NHibernate .dlls to the application
2. Configure NHibernate so that you can create session factory and session
3. Map your database tables with your application objects (class)

Let us discuss each of the steps in detail. We are just discussing the requirements and techniques here; we will try to create a c# project at the end of the article.

**Install NHibernate**

You can easily download the latest version of **NHibernate** from their official site. Just unzip the package and add reference of the required dlls. Normally you will require “NHibernate.dll” and “Iesi.Collections.dll”. The best approach and you should use the NuGet package manager to download and install NHibernate.

**Configuring NHibernate and create Session**

To configure and create NHibernate session we have to perform following steps:

1. Create an instance of NHibernate.Cfg.Configuration class
2. Providing custom configuration parameter if required
3. Create instance of ISessionFactory using configuration instance
4. Create instance of ISession using ISessionFactory instance

**NHibernate.Cfg.Configuration**

The instance of “NHibernate.Cfg.Configuration” class holds all the mapping between .net objects and database objects (we will discuss later how to provide the mapping to the configuration). The configuration instance uses various configuration parameters. NHibernate provides a lots of configuration parameters to support many different environments. All the configuration parameter has their optional value. But we can also customize the configuration value according to our needs.

Creating instance is as simple as creating instances of any of the .net classes:

var cfg = new NHibernate.Cfg.Configuration();

or if we use “NHibernate.Cfg” at using directives then simply

var cfg = new Configuration();

**Providing custom configuration parameter**

In the following link at “**Table 3.1**” you will find a list of **NHibernate ADO.NET Configuration Properties**: <http://nhibernate.info/doc/nh/en/index.html#configuration-hibernatejdbc> And In the following link at “**Table 3.2**” you will find a list of some of the **NHibernate Optional Configuration Properties**: <http://nhibernate.info/doc/nh/en/index.html#configuration-optional>

As we already discussed that NHibernate provides default value for almost all of the configurations, still we may need to customize some of them. Most of the general cases we need to update following properties:

1. NHibernet Dialect: We need to provide dialect according to our database. Following link at “**Table 3.3**” provides a list of dialect that NHibernet supports for different databases : <http://nhibernate.info/doc/nh/en/index.html#configuration-optional-dialects>
2. Connection string: We need to provide our own database connection string

We can provide custom configuration value using following three ways:

1. App/web config file
2. Separate XML configuration file and
3. Programmatically (by the code)

Let us see them in examples:

1. App/web config file: We have to add a separate configsection of type “NHibernate.Cfg.ConfigurationSectionHandler” to provide the configuration parameter. Following is an example

﻿<?xml version="1.0" encoding="utf-8" ?>

<configuration>

<configSections>

<section name="hibernate-configuration" type="NHibernate.Cfg.ConfigurationSectionHandler, NHibernate" />

</configSections>

<hibernate-configuration xmlns="urn:nhibernate-configuration-2.2">

<session-factory>

<property name="dialect">NHibernate.Dialect.MsSql2012Dialect</property>

<property name="connection.connection\_string\_name">DBConnection</property>

<mapping assembly="LearningNH" />

</session-factory>

</hibernate-configuration>

<connectionStrings>

<add name="DBConnection" connectionString="Data Source=BULBUL\MSSQLSERVER12;Initial Catalog=EmployeeDB;Trusted\_Connection=True;"/>

</connectionStrings>

</configuration>

1. XML configuration file: We use a separate xml file named “hibernate.cfg.xml” to specify the configuration parameter. This file should be resided to the application directory otherwise we need to specify the file path at the code. Following is an example

<?xml version='1.0' encoding='utf-8'?>

<hibernate-configuration xmlns="urn:nhibernate-configuration-2.2">

<!-- an ISessionFactory instance -->

<session-factory>

<!-- properties -->

<property name="connection.provider">NHibernate.Connection.DriverConnectionProvider</property>

<property name="connection.driver\_class">NHibernate.Driver.SqlClientDriver</property>

<property name="connection.connection\_string">Server=localhost;initial catalog=nhibernate;User Id=;Password=</property>

<property name="show\_sql">false</property>

<property name="dialect">NHibernate.Dialect.MsSql2000Dialect</property>

<!-- mapping files -->

<mapping resource="NHibernate.Auction.Item.hbm.xml" assembly="NHibernate.Auction" />

<mapping resource="NHibernate.Auction.Bid.hbm.xml" assembly="NHibernate.Auction" />

</session-factory>

</hibernate-configuration>

1. Programmatically : We can set configuration properties using “SetProperty” method of “Configuration” instance. This method takes name and value of the configuration as parameter. For example

cfg.SetProperty(NHibernate.Cfg.Environment.ShowSql, "true");

cfg.SetProperty(NHibernate.Cfg.Environment.FormatSql, "false");

cfg.SetProperty(NHibernate.Cfg.Environment.Dialect, "NHibernate.Dialect.MsSql2012Dialect");

var connectionString = System.Configuration.ConfigurationManager.ConnectionStrings[“DbConnection”]

cfg.SetProperty(NHibernate.Cfg.Environment.ConnectionString, connectionString);

**ISessionFactory and ISession**

Before discussing how to create ISessionFactory and ISession, let us discuss what they are in NHibernate. We use ISession to perform all the CRUD operations and transaction management. An ISession is usually used as a single unit of work. It is not thread safe. We should not share it within different threads. We should open a session only for a single unit of business process and close (discard) the session after completing the process. We create ISession from ISessionFactory.

ISessionFactory is thread safe (immutable). We can share it within all the threads of our application. Creating ISessionFactory is expensive so it is best practice to create a single instance of ISessionFactory for each of the database, store it and use when required. It is good idea to use thread safe single tone pattern to create ISessionFactory. Following link describes a thread safe single tone pattern:

<https://msdn.microsoft.com/en-us/library/ff650316.aspx>

Session management is a very important strategy we should take care of. It is different for different type of applications (like web, desktop, web services etc.). But we will not discuss session management strategy in this article.

**Creating ISessionFactory**

Before creating IsessionFactory we need to add all the mapping files to the configuration. How to create mapping file, we will discuss later but for now be informed that the mapping file is an xml file with extension “.hbm.xml” that contains mapping information between database tables and business objects (classes).

**Adding mapping files to the configuration**

There are various ways to add the mapping files like,

1. We can add xml file directly, cfg.AddFile("Item.hbm.xml")
2. We can add .net class associated with mapping file, cfg.AddClass(typeof(Item))//Item is a .net class type
3. We can add stream of xml file, cfg.AddInputStream(memorystream)// memorystream is a stream of mapping file
4. We can add the name of the assembly, cfg.AddAssembly(“LearningNH”) //LearningNH is the name of the assembly

Probably the best option is to add the assembly name. NHibernate will find the mapping files from the embedded resources. In this case we need to set the “build action” property of the mapping xml files to “Embedded Resources”.

When all the mapping is parsed by the configuration, creating ISessionFactory” is as simple as just calling a method of the instance of configuration class (“NHibernate.Cfg.Configuration”). Following is the example of creating “ISessionFactory”:

ISessionFactory sessionFactory = cfg.BuildSessionFactory();

As we already discussed that creating ISessionFactory is expensive, so we should use single ton pattern to create ISessionFactory.

**Creating ISession**

We need to call a method of ISessionFactory to open a new session, like

ISession session = sessionFactory.OpenSession();

We can also provide a custom ADO.Net connection as a parameter of “OpenSession" method.

As we already discussed session management strategy is different for different type of application, we need to choose appropriate strategy for our application. But we will not discuss about session management strategy in this article.

**Mapping Database Tables with Business Objects (Class)**

As we already discussed that, we need to provide mapping information among the .net objects and database tables to nhibernate configuration before creating ISesssionFactory. Following are ways we can define mapping information:

1. Fluent NHibernate
2. Mapping by code
3. By XML file.

In this article we will only discuss about mapping by xml file.

**Mapping Example between Employee Class and Employee Table**

Let us consider that we have a class Employee and corresponding database table Employee.

Employee class example:

public class Employee

{

public virtual int EmployeeId { get; set; }

public virtual string EmployeeNo { get; set; }

public virtual string Name { get; set; }

public virtual bool IsMarried { get; set; }

public virtual DateTime DateOfBirth { get; set; }

}

And the sql of creating database table is:

CREATE TABLE Employee

(

employee\_id int NOT NULL,

employee\_no varchar(10) NOT NULL,

employee\_name varchar(255),

marital\_statue tinyint,

birth\_date DateTime,

CONSTRAINT pk\_Employee PRIMARY KEY (employee\_id)

)

Following is an example of mapping file between above class and table:

﻿<?xml version="1.0" encoding="utf-8" ?>

<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2" assembly="LearningNH" namespace="LearningNH" schema="EmployeeDB.dbo">

<class name="Employee" table="Employee">

<id name="EmployeeId" column="employee\_id" generator="identity" />

<property name="EmployeeNo" column="employee\_no" />

<property name="Name" column="employee\_name" />

<property name="IsMarried" column="marital\_statue" />

<property name="DateOfBirth" column="birth\_date" />

</class>

</hibernate-mapping>

Let us discuss the mapping some detail. The first tag of the file is “<hibernate-mapping>”. It has several optional attributes like:

1. schema: The name of the DataBase schema from where the tables will be mapped.
2. assembly: The name of the assembly from where the classes will be mapped.
3. namespace: The namespace of the class, if we don’t use this attribute we have to write fully qualified class name.

In the following link you will find other optional attributes of “<hibernate-mapping>” tag:

<http://nhibernate.info/doc/nh/en/index.html#mapping-declaration-mapping>

Within the “<hibernate-mapping>” tag, we need to declare <class> element. <class> tag requires following attributes:

1. name: Name of the class. If namespace is not declared then fully qualified name
2. table: Name of the database table. If class and database table has same name, then we can omit this.

In the following link you will find other optional attributes of “<class>” tag:

<http://nhibernate.info/doc/nh/en/index.html#mapping-declaration-class>

We can define multiple <class> elements within same “<hibernate-mapping>” tag, in a single file, but it is better to use separate xml file for each of the classes.

Within the <class> tag we have to define the mapping among the database columns and classes property. We need to use <id> tag to map primary key column and for other column we need to use <property> tag.

**ID Mapping**

<id> element defines the mapping between **primary key column** of the database table and property of the .net class. It has following attribute:

1. name: Name of the property of the class
2. column: Name of the column of the table, if class property and column name is same we can omit this.
3. generator: Name of the .net class implemented by nhibernate(we can provide our own also).

“generator ” is very important attribute. It selects the .net class to generate the unique identifier for the primary key column. If any parameter requires for the “generator” class then we need to declare it within a separate “<generator>” element and define the required parameter as “<param>”. Following is an example:

<generator class="NHibernate.Id.TableHiLoGenerator">

<param name="table">uid\_table</param>

<param name="column">next\_hi\_value\_column</param>

</generator>

In the following link you will find the name and description about nhibernate provided generator classes:

<http://nhibernate.info/doc/nh/en/index.html#mapping-declaration-id-generator>

**Property Mapping**

<property> element defines the mapping between **non primary key column** of the database table and property of the .net class. Like <id> tag it also has “name” and “column” attribute but no “generator” attribute. In the following link you will find other optional attributes of “<property>” tag:

<http://nhibernate.info/doc/nh/en/index.html#mapping-declaration-property>

**Relational Mapping**

In general, relational database defines following types of relationships:

1. One to one
2. One to many
3. Many to one and
4. Many to many

Let us discuss how to map each of the relationships by nhibernate.