

Tales from the Evil Empire

Bertrand Le Roy's blog

A total n00b's guide to migrating from a custom data layer to Nhibernate: getting started



(Screencast can be found at the end of the post)

To be clear when I say "total n00b", I'm not talking about you, dear reader, I'm talking about me. The last time I wrote any serious data access code was *circa* 2002. Since then, I got hired by the Evil Empire and started developing new tools to make it easier to build demos of Northwind master-details. I jest, I jest. Or do I?

So let me explain what I'm going to talk about in this and future related blog posts. We have [this e-commerce application that Rob started](#) and that we're going to continue developing. Last time Rob touched the data access, he wanted to experiment with going back to less abstraction and to working directly with that interesting data-centric Domain Specific Language, you know, SQL. So he went ahead and played with that T4-driven DAL generator. It was interesting as an experiment, and Rob's blog series is all about experimenting publicly, but let's face it, that data access didn't fly much farther than that. To be clear, I'm not saying that Rob was to blame for our sucky data access layer, but the truth is that we have had this Franken-DAL from the nineties in the code for a few months now and it needs replacing. So what do we do?

First, we look around and find out what other people out there are doing. We could go for the Microsoft flavor of ORM, EF, but we're not going to do that (at least not yet) and will instead go for the one that is the most widely used by the community as of today, and that is, I believe, [NHibernate](#). The home page for the project itself can be found at <http://nhforge.org> and the best place to get started is probably the tutorial section of that site: <http://nhforge.org/doc/nh/en/index.html#quickstart>.

In this first post, I'll just play with the library and try to get a list of products to display on a page. In future posts, I'll look at the actual app and start migrating it. For this time, because I'm just trying to get to grips with a library I don't know, it's going to be quick and it's going to be dirty. Definitely don't take any of what you are about to see as best practices. Ever.

The first step is to add the NHibernate and dependencies to the project. The library can be downloaded from SourceForge (<http://sourceforge.net/projects/nhibernate>). Nhibernate is itself under [LGPL](#) but it comes with the following dependencies:

- [Antlr](#): a library to construct grammar parsers, interpreters, etc. typically for domain specific languages. It's under a [BSD license](#).
- [Iesi.Collections](#): a library that implements sets (collections with unique elements) and that doesn't have a definite license. A [comment from the author](#) seems to implicate public domain.
- [Log4net](#): the leading logging library for .NET, licensed under [Apache 2.0](#) and managed by the [Apache Foundation](#).

You may not care too much about that, but your boss and his lawyer might... Technically, all you need to add to the project is a reference to the NHibernate dll and the others will follow.

Once the reference has been added to our project, we can start configuring. This can be done in a variety of manners but the easiest is to do it through web.config. The basic configuration for NHibernate consists in the connection string, the SQL dialect and the proxy factory to use. Because we don't want to repeat ourselves and because I prefer my connection strings to live in the connection strings section of web.config, I'll use a "connection.connection_string_name" setting instead of a "connection.connection_string" setting:

```
<hibernate-configuration xmlns="urn:nhibernate-configuration-2.2">
  <session-factory>
    <property name="connection.connection_string_name">
      KonaConnectionString
    </property>
    <property name="dialect">
      NHibernate.Dialect.MsSql2000Dialect
    </property>
    <property name="proxyfactory.factory_class">
```

```

    NHibernate.ByteCode.LinFu.ProxyFactoryFactory,
    NHibernate.ByteCode.LinFu
  </property>
</session-factory>
</hibernate-configuration>

```

The proxy factory factory class stuff looks a little intimidating and a little over-architected from just looking at it (a factory factory? To create proxies?) but it's not as bad as it looks. What this is doing is declaring what library to use to generate dynamic proxies for our data classes. It is a good thing that Nhibernate is open to multiple providers here, and the good news is this is probably the first and last time you need to know about this. Just make a choice if you care (and know why you care), or use this one (licensed under LGPL) if you don't.

But what is a dynamic proxy, you may ask? First and foremost, it is a proxy class that will be used in place of the POCO class that you will provide to represent your data. Its purpose is to intercept all calls into the object's properties, both getters and setters, and to handle things like lazy loading. The dynamic aspect of it is that those proxy classes are dynamically generated, usually using Reflection.Emit. This means that until .NET 4.0 is here, using lazy loading in Nhibernate will prevent the application from running in medium trust. More on that in future posts, but know that lazy loading is not mandatory and can easily be disabled from the mapping file.

The next thing to do is to grab the Nhibernate configuration in order to create a session out of it. The configuration can be built from the web.config data like this:

```
var nhconfig = new NHibernate.Cfg.Configuration().Configure();
```

A session is the mediator between your code and Nhibernate. According to the documentation, it is a "short-lived object representing a conversation between the application and the persistent store". Its lifetime is usually the same as the lifetime of the request. Here's how you can create a session:

```
var session = nhconfig.BuildSessionFactory().OpenSession();
```

With all that we are pretty much set-up but we are still lacking any data. Let's fix that and attempt our first mapping. I'm going to use a simplified version of the product database we have in the commerce app (which is none other than AdventureWorks). In there, I have a Products table that looks like this:

	Column Name	Data Type	Allow Nulls
🔑	SKU	nvarchar(50)	<input type="checkbox"/>
	SiteID	uniqueidentifier	<input type="checkbox"/>
	DeliveryMethodID	int	<input type="checkbox"/>
	ProductName	nvarchar(50)	<input type="checkbox"/>
	BasePrice	decimal(18, 0)	<input type="checkbox"/>
	WeightInPounds	money	<input type="checkbox"/>
	DateAvailable	datetime	<input type="checkbox"/>
	InventoryStatusID	int	<input type="checkbox"/>
	EstimatedDelivery	nvarchar(50)	<input type="checkbox"/>
	AllowBackOrder	bit	<input type="checkbox"/>
	IsTaxable	bit	<input type="checkbox"/>
	DefaultImageFile	nvarchar(255)	<input checked="" type="checkbox"/>
	Version	timestamp	<input type="checkbox"/>
	AmountOnHand	int	<input type="checkbox"/>
	AllowPreOrder	bit	<input type="checkbox"/>
	Options	image	<input checked="" type="checkbox"/>

We don't have to map everything in that table so we won't (yet). Instead, we'll create a simple and incomplete class to represent a product:

```

public class Product {
    public virtual string Sku { get; set; }
    public virtual Guid SiteID { get; set; }
    public virtual string Name { get; set; }
    public virtual double BasePrice { get; set; }
}

```

}

Notice how all fields are virtual here. This, again, is done so that a dynamic proxy class that overrides those properties can be created.

Now we have the O (object) and the R (relational), so let's do the M. The mapping is an XML file that we'll put next to our Product class:

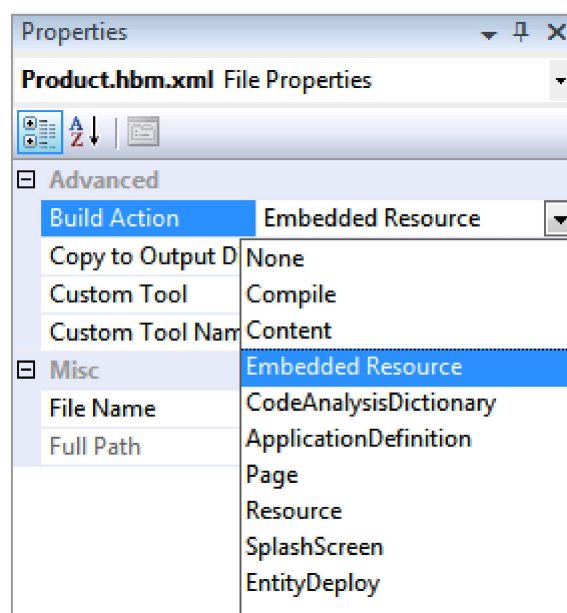
```
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-mapping xmlns="urn:hibernate-mapping-2.2"
  namespace="Nhib.Models" assembly="Nhib">

  <class name="Product" table="Products">
    <id name="Sku">
      <column name="SKU" sql-type="nvarchar(50)" not-null="true"/>
      <generator class="uuid.hex" />
    </id>

    <property name="SiteID"/>
    <property name="Name">
      <column name="ProductName"/>
    </property>
    <property name="BasePrice" />
  </class>

</hibernate-mapping>
```

The mapping class is named like the Product class, with a ".hbm.xml" extension. It is also configured to build as an embedded resource so that Hibernate can reflect on it at runtime. This is done by setting the Build Action in the properties of the file in Project Explorer:



And we're done as far as setup is concerned. Now all that remains to do is to actually query and display that data.

Because I'm just playing with the framework at this point, I'll just query from the Index controller action:

```
var nhconfig = new NHibernate.Cfg.Configuration().Configure();
nhconfig.AddClass(typeof(Product));
using (var session = nhconfig.BuildSessionFactory().OpenSession()) {
    var query = session.CreateQuery("select p from Product as p");
    ViewData.Model = query.List<Product>();
}
```

UPDATE: Ayende is right to attract my attention to the fact that building a session factory is an expensive operation and so you should treat it as a singleton. In other words, create the factory once and store it in a static variable. For the sake of the simplicity of the example, I'm leaving the code above, but again, don't do this at home...

And finally, let's render this from the view:

```
<ul>
<% foreach (var p in Model) { %>
```

```
<li><%= Html.Encode(p.Name) %> - $<%= p.BasePrice %></li>
<% } %>
</ul>
```

The results look like this:

- Hiking Backpack - \$50
- Wide-base Backpack - \$50
- Short Backpack - \$40
- Mountaineering Backpack - \$130
- Sprint 500 Bike - \$460
- Escape 3.0 Bike - \$680
- Scoop Cruiser - \$380
- Adventure Works 20x50 Binoculars - \$170
- Adventure Works 20x30 Binoculars - \$170
- Sierra Leather Hiking Boots - \$90

I hope this gives a good idea of how simple it is to get started with NHibernate, because it really is. I knew near to nothing about it and almost didn't have to fight the framework to get it running. Seeing the first successful rendering on my very first CTRL+F5 was very encouraging.

In a future post, I'll show how this applies to a real-world application and how we migrated such an application from its existing data layer to one that uses NHibernate.

The source code can be downloaded from here: [NHibGettingStarted.zip](#) (warning: all files provided under their respective licenses).

... and here's the screencast:

Posted: [Aug 17 2009, 04:29 PM](#) by [Bertrand Le Roy](#) | with [21 comment\(s\)](#)
Filed under: [ASP.NET](#), [MVC](#), [NHibernate](#), [Screencast](#)

Comments

joechung said:

I have a question not necessarily just for you but for the community at large: why is Nhibernate so popular?

August 17, 2009 9:29 PM

Troy Goode said:

I highly recommend taking a gander at Fluent Nhibernate - it manages to replace all that XML yuckiness with a beautiful fluent interface. Your Product.hbm.xml file could be replaced by something that looks a little like:

```
public ProductMap : ClassMap<Product>{  
  
    Id(x=> x.Sku, "SKU").Length(50).NotNullable();  
  
    Map(x=> x.SiteID);  
  
    Map(x=> x.Name, "ProductName");  
  
    Map(x=> x.BasePrice);  
  
}
```

<http://fluentnhibernate.org/>

August 17, 2009 10:50 PM

Andrei Rinea said:

Why on earth isn't there a visual tool to set up the mapping(s) for Nhibernate? And to offtopic it a little, the same goes for the configuration of WCF services too..

August 18, 2009 2:24 AM

Andrew Rea said:

WCF Does have a GUI for the configuration. I am almost sure Nhibernate has a query generator and I would not be surprised if there was also a GUI for the configuration BUT, inside Visual Studio, when on the mapping file, if you right click and properties you are able to add the schemas to the XML file, and this allows for autocomplete to popup and enables you to have a wander round the different attributes and learn more!

Another one to look out for is using a version of the UnitOfWork pattern, one of the them I use is the NhibernateUnitOfWork. I would also point out that you do not have to physically add the class to the configuration as it will automatically scan a location for these providing you state what assembly to use!!

: -)

Andrew

August 18, 2009 3:32 AM

Mo said:

why not LINQ?

August 18, 2009 11:34 AM

Dave said:

Would any sizable site choose Nhibernate over custom data access code? Is the answer yes to all of the following.

1. I want to know nothing about how my data is stored.
2. I will never need to write reports.
3. I will deal with scaling at a later time.

August 18, 2009 12:38 PM

Adam Aldrich said:

@Andrei Rinea NHibernate is open source and you are welcome to develop one :)

Most developers that use NHibernate (myself included) prefer not to have a visual designer. It just doesn't scale well in an enterprise application where you have lots of business entities.

I think you will find that using Fluent interfaces or even mapping files is really not bad and works better than the draggy droppy interfaces Linq to Sql or Entity Framework push on you.

August 18, 2009 1:08 PM

lucasbfr said:

@Dave: I completely disagree, the SQL generated by Hibernate is predictable, and can be overridden when necessary. Not using that kind of framework is most of the times a waste of time.

@Bertrand: Did you have a look at Castle's ActiveRecord framework? It encapsulates nHibernate and avoids the pain of maintaining xmls: www.castleproject.org/.../index.html

August 18, 2009 2:11 PM

Michael Sanders said:

So, (at least from this example), it appears you still have to have very intimate knowledge of the data tables. Is there not some tool, that can scan the database in question, and pre-populate a lot of these mappings?

August 18, 2009 3:43 PM

Eyston said:

Dave:

1 - What do you mean? You can define DB first and NH second (its what I have to do working with existing ERP system).

2 - Do reporting outside of NH, it isn't a reporting tool.

3 - NH gives me a smart and distributed cache from the get go. Maybe you are a very good programmer, but it would take me a lot longer to engineer that cache myself in a custom system. Not doing a query is going to be faster than a fine tuned query.

Also, if you want to fine tune queries, you can. I haven't needed to.

But I think focusing on queries is missing the point. It is like becoming the best at using a slide rule when calculators are invented. If you want to scale you need to think at a higher level than NH.

August 18, 2009 3:47 PM

Joe Chung said:

@lucasbfr, Castle's ActiveRecord looks interesting. How does it compare to Subsonic?

August 18, 2009 7:23 PM

Greg-Muellerson said:

Hey, ok, I get it, I guess - but does this really work?

August 18, 2009 8:41 PM

Vladimir said:

Castle's ActiveRecord is still in development phase. The latest release is 1.0 Release Candidate 2.

However, it is a very nice way to avoid XML files.

August 19, 2009 4:04 AM

Lee Timmins said:

I prefer to use T4 templates to generate my entities and mappings. One thing i don't like about Nhibernate is that i'm not a big fan of the query language and would prefer to do it with linq. Bertrand do you have plans to implement the model with linq to entities once v4 comes out?

August 19, 2009 6:18 AM

jbland said:

Just FYI, but as of 2.1, Nhibernate can run in medium trust - see blogs.taiga.nl/.../new-adventures-under-medium-trust

August 19, 2009 10:23 AM

Andrew said:

How would you access multiple databases with Nhibernate? We have some data in Oracle and some in SQL Server.

August 19, 2009 5:53 PM

Matt said:

@Bertrand: thank you very much for this introduction. It gave me a great overview of what it is all about!

Looking forward to your next articles about Nhibernate!

Matt

August 19, 2009 6:14 PM

Fern said:

I have used Nhib and I've use EntitySpaces (which is not Entity Framework from MS). I have to say EntitySpaces is way easier to setup and develop in. It's a VS plugin that you just point to a database and it generates everything you need to start working with data. No xml files to mess with, you don't have to type out the model. Check out the screencast off their home page... www.entityspaces.net/.../Default.aspx

August 19, 2009 10:18 PM

Dave R. said:

@Fern - EntitySpaces is a commercial product (from \$300). I'd certainly expect autogen when I have to spend that sort of money!

August 20, 2009 7:07 AM

lucasbfr said:

@Joe Chung: Unfortunately, Castle ActiveRecord was a technical choice before I came on my current project so I haven't had the opportunity to look at Subsonic closely.

AR is built on top of NHibernate, which made it more appealing at the time.

(Fluent NHibernate looks nice too)

August 20, 2009 9:01 AM

Eyston said:

@Lee Timmins: NH Linq is released. It lets you do Linq syntax (from Bertrand's query):

```
var products = from p in session.Linq<Product>() select p;
```

```
ViewData.Model = products.ToList();
```

@Andrew: Each session factory is responsible for a database, so you would need multiple factories.

August 20, 2009 3:17 PM

[Terms of Use](#)