PROGRESS® DataDirect®

WHITE PAPER

www.progress.com

GETTING STARTED WITH 100% MANAGED CODE ACCESS TO ORACLE FROM NHIBERNATE

PROGRESS

TABLE OF CONTENTS

INTRODUCTION

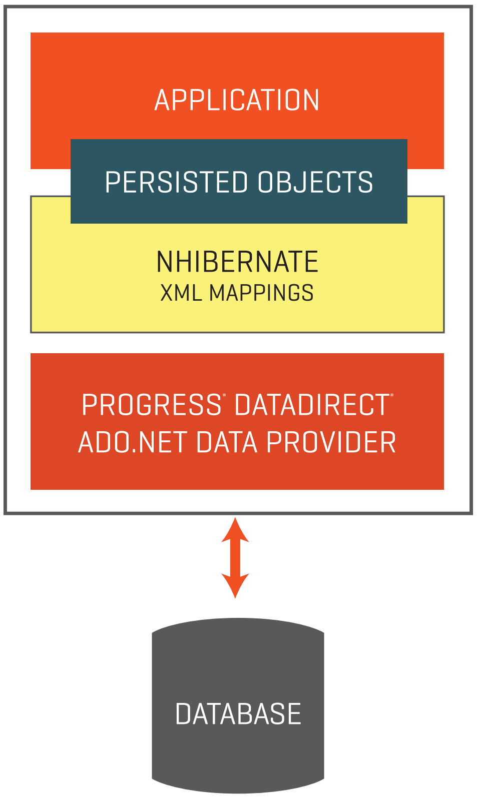
NHibernate is a mature, open source, object-relational mapper for the .NET framework. It allows you to easily store objects in a database by automatically generating SQL-based on XML metadata files. NHibernate works with any C# class and does not require that the objects implement a special interface or derive from a common-base class.

In this tutorial, we will walk through the following steps for creating a simple NHibernate project to connect with Oracle with an IDriver class using a 100% managed code ADO.NET data provider:

- Creating a Visual Studio C# project
- Adding the required Oracle and NHibernate references
- Creating the IDriver class for the Oracle provider
- Creating the Domain class diagrams, Domain classes, and mapping files
- Creating a repository class to fetch and store objects
- And more

We will assume that you already have a suitable version of Visual Studio NHibernate installed. We will be using Visual Studio Team System 2008 and NHibernate 2.1.2.

This example uses the Progress® DataDirect Connect® *for* ADO.NET data providers with NHibernate. All Progress DataDirect ADO.NET providers for Oracle, Sybase, DB2, and SQL Server are 100% managed code. Unlike other providers, which require database client libraries to support some or all of their features, true 100% managed code ADO.NET data providers never need database client libraries such as Oracle SQL*Net. In addition to eliminating the need to download, install, configure, and deploy database client libraries, 100% managed code architecture also translates to superior application performance and better application security for NHibernate applications of all types.



NOTE: This article was written with release 3.3 of the DataDirect Connect *for* ADO.NET Oracle provider. If you are using a more current release, you should be able to follow these same steps as well, although there may be some minor differences from release to release.

You can download the latest release of the DataDirect Connect *for* ADO.NET providers from: http://web.datadirect.com/docs/gated/software-download/connect-ado-net.html

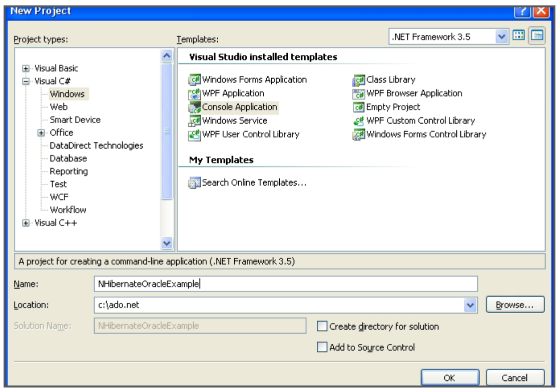
Instructions for installing the Oracle provider are available from: http://web.datadirect.com/docs/public/download/eval-docs/adonet-oracle.pdf

PROGRESS

INSTRUCTIONS

1. Launch Visual Studio and create a new Visual C# Console Application project called NHibernateOracleExample.

2. Right-click the References folder in the SolutionExplorer and select "Add Reference..." from the context menu. We need to add the NHibernate, NHibernate.ByteCode.Castle and DDTek.Oracle references. If the references are not listed in the .NET tab of the Add Reference dialog, select the Browse tab and locate the appropriate NHibernate.dll, NHibernate.ByteCode.Castle.dll, and DDTek.Oracle.dll assemblies. Note that the NHibernate.ByteCode.Castle assembly is not a required reference at compile time, but adding it as a reference causes Visual Studio to copy it to the Output directory so it will be found at runtime.



New Project

Project types: Visual Basic, Visual C# (Windows, Web, Smart Device, Office, DataDirect Technologies, Database, Reporting, Test, WCF, Workflow), Visual C++

Templates: .NET Framework 3.5

Visual Studio installed templates: Windows Forms Application, WPF Application, Console Application, Windows Service, WPF User Control Library, Class Library, WPF Browser Application, Empty Project, WPF Custom Control Library, Windows Forms Control Library

My Templates: Search Online Templates...

A project for creating a command-line application (.NET Framework 3.5)

Name: NHibernateOracleExample

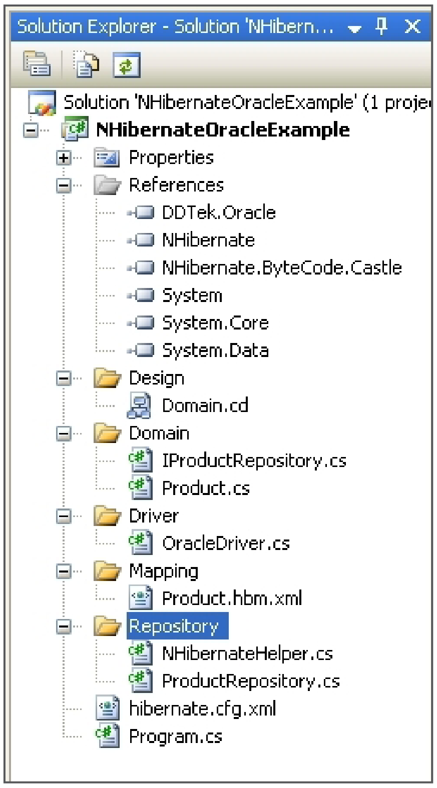
Location: c:\ado.net

Solution Name: NHibernateOracleExample

Create directory for solution

Add to Source Control

3. Create the following folders in your project: Domain, Design, Mapping, Repository, and Driver. We will use the folders to organize various metadata files and helper classes. Here is a sneak peak at what the project structure will look like when finished:



Solution Explorer - Solution 'NHibern...

```
Solution 'NHibernateOracleExample' (1 proje
  NHibernateOracleExample
    Properties
    References
      DDTek.Oracle
      NHibernate
      NHibernate.ByteCode.Castle
      System
      System.Core
      System.Data
    Design
      Domain.cd
    Domain
      IProductRepository.cs
      Product.cs
    Driver
      OracleDriver.cs
    Mapping
      Product.hbm.xml
    Repository
      NHibernateHelper.cs
      ProductRepository.cs
    hibernate.cfg.xml
    Program.cs
```

4. Add a new class file to the Driver folder called OracleDriver.cs with the following contents:

```
using System;
using System.Data;
using DDTek.Oracle;
using NHibernate.Driver;

namespace NHibernateOracleExample.Driver {
  public class OracleDriver : DriverBase {
    public override IDbCommand CreateCommand() {
      return new OracleCommand();
    }
    public override IDbConnection CreateConnection() {
      return new OracleConnection();
    }
    public override string NamedPrefix {
      get { return string.Empty; }
    }
    public override bool UseNamedPrefixInParameter {
      get { return false; }
    }
    public override bool UseNamedPrefixInSql {
      get { return false; }
    }
  }
}
```

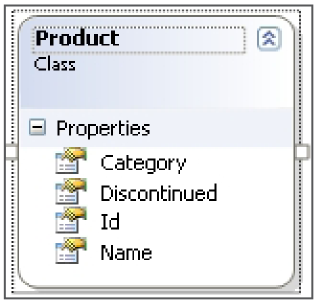
This class derives from the NHibernate.Driver.DriverBase base class and allows NHibernate to hook into the Oracle provider. You could also implement this class in a separate Code Library project and add a reference to it since it will likely be reused across many different projects.

5. Add a new class to the Domain folder called Product.cs with the following contents:

```
using System;
namespace NHibernateOracleExample.Domain {
  class Product {
    public virtual Guid Id { get; set; }
    public virtual string Name { get; set; }
    public virtual string Category { get; set; }
    public virtual bool Discontinued { get; set; }
  }
}
```

This is the object that will be stored in the database. Notice that the class does not derive from a special class or implement a special interface. Also notice, however, that we must define each of the properties in the class as virtual. This is a NHibernate requirement.

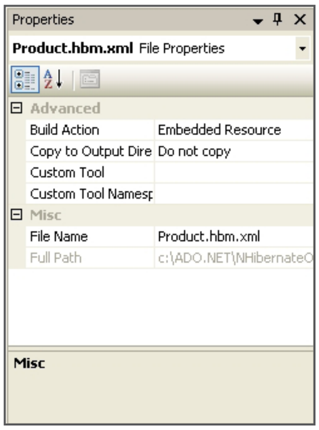
6. Add a new Class Diagram called Domain.cd to the Design folder. This diagram will help to visualize our domain. Drag the Product class from the Class View to add it to the design. Note that this step is not required but is a recommended NHibernate best practice as part of Domain Driven Design.



7. Add a new XML file called Product.hbm.xml to the Mapping folder as an embedded resource with the following contents:

```
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2"
                assembly="NHibernateOracleExample"
                namespace="NHibernateOracleExample.Domain">
  <class name="Product">
    <id name="Id">
      <generator class="guid" />
    </id>
    <property name="Name" />
    <property name="Category" />
    <property name="Discontinued" />
  </class>
</hibernate-mapping>
```

This XML file defines the mappings for the Product class. Notice that no datatype information for the properties is given. NHibernate will use reflection to determine appropriate datatypes. Remember to set the Build Action to Embedded Resource.


Properties

Product.hbm.xml File Properties

Advanced	
Build Action	Embedded Resource
Copy to Output Dire	Do not copy
Custom Tool	
Custom Tool Namesp	
Misc	
File Name	Product.hbm.xml
Full Path	c:\ADO.NET\NHibernateO

Misc

8. Add a new interface called IProductRepository to the Domain folder with the following contents:

```
using System;
using System.Collections.Generic;

namespace NHibernateOracleExample.Domain {
  interface IProductRepository {
    void Add(Product product);
```

PROGRESS

```
    void Update(Product product);
    void Remove(Product product);
    Product GetById(Guid productId);
    Product GetByName(string name);
    ICollection<Product> GetByCategory(string category);
  }
}
```

Defining an interface for these operations keeps the design of the domain and the actual nuts-and-bolts of how they are implemented separate.

9. Add a new class called NHibernateHelper.cs to the Repository folder with the following contents:

```
using System;
using NHibernateOracleExample.Domain;
using NHibernate;
using NHibernate.Cfg;

namespace NHibernateOracleExample.Repository {
  public class NHibernateHelper {
    private static ISessionFactory _sessionFactory;
    private static ISessionFactory SessionFactory {
      get {
        if (_sessionFactory == null) {
          var configuration = new Configuration();
          configuration.Configure();
          configuration.AddAssembly(typeof(Product).Assembly);
          _sessionFactory = configuration.BuildSessionFactory();
        }
        return _sessionFactory;
      }
    }
    public static ISession OpenSession() {
      return SessionFactory.OpenSession();
    }
  }
}
```

This class creates a new session for the caller. It maintains a reference to an ISessionFactory class so that it isn't recreated each time a session is needed.

10. Add a new class to the Repository folder called ProductRepository.cs with the following contents:

```
using System;
using NHibernate;
using NHibernate.Criterion;
using NHibernateOracleExample.Domain;
```

```
namespace NHibernateOracleExample.Repository {
  class ProductRepository : IProductRepository {
    public void Add(Product product) {
      using (ISession session = NHibernateHelper.OpenSession()) {
        using (ITransaction transaction = session.
BeginTransaction()) {
          session.Save(product);
          transaction.Commit();
        }
      }
    }
    public void Update(Product product) {
      using (ISession session = NHibernateHelper.OpenSession()) {
        using (ITransaction transaction = session.
BeginTransaction()) {
          session.Update(product);
          transaction.Commit();
        }
      }
    }
    public void Remove(Product product) {
      using (ISession session = NHibernateHelper.OpenSession()) {
        using (ITransaction transaction = session.
BeginTransaction()) {
          session.Delete(product);
          transaction.Commit();
        }
      }
    }
    public Product GetById(Guid productId) {
      using (ISession session = NHibernateHelper.OpenSession()) {
        return session.Get<Product>(productId);
      }
    }
    public Product GetByName(string name) {
      using (ISession session = NHibernateHelper.OpenSession()) {
        Product product = session
          .CreateCriteria(typeof(Product))
          .Add(Restrictions.Eq("Name", name))
          .UniqueResult<Product>();
        return product;
      }
    }
    public System.Collections.Generic.ICollection<Product>
GetByCategory(string category) {
      using (ISession session = NHibernateHelper.OpenSession()) {
      var products = session
          .CreateCriteria(typeof(Product))
```

```
                .Add(Restrictions.Eq("Category", category))
                .List<Product>();
              return products;
            }
          }
        }
      }
```

This is the NHibernate-specific implementation of the IProductRepository interface. Each of the methods creates a new session and uses it to perform the requested action. By implementing a repository class, the code that handles the business logic can be naïve about the specifics of CRUD operations.

11. We are finally ready to write the code that calls the classes just implemented. Add the following using statements to the Program.cs file:

```
using System;
using NHibernate;
using NHibernate.Cfg;
using NHibernate.Tool.hbm2ddl;
using NHibernateOracleExample.Domain;
using NHibernateOracleExample.Repository;
```

Now add the following code to the Main method of Program.cs:

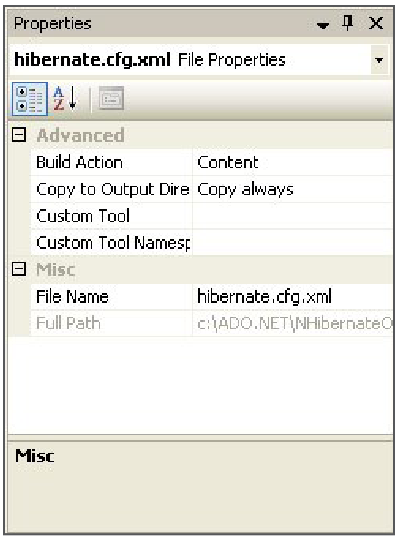
```
  var cfg = new Configuration();
  cfg.Configure();
  cfg.AddAssembly(typeof(Product).Assembly);

  SchemaExport se = new SchemaExport(cfg);
  se.Execute(false, true, false);

  IProductRepository repository = new ProductRepository();
```

The code above creates a new Configuration instance and uses the AddAssembly method to load the embedded *.hbm.xml resource. Remember that the Product.hbm.xml file (which we embedded as a resource) contains the mappings for our Product class. Next, a new SchemaExport instance is created that is used to automatically create the database tables to store Product objects. And finally create a
new IProductRepository, which will allow us to perform insert and fetch operations.

12. Finally, create a new Product instance and add it to the database. Add the following lines to the bottom of the Main method:

```
var product = new Product { Name = "Apple", Category =
"Fruits" };
  repository.Add(product);
```

We've created a new Product instance and used the repository to add it to the database.



Advanced	
Build Action	Content
Copy to Output Dire	Copy always
Custom Tool	
Custom Tool Namesp	
Misc	
File Name	hibernate.cfg.xml
Full Path	c:\ADO.NET\NHibernateO

13. If you attempt to run the project as is, you'll likely run into issues since we haven't yet added our configuration file, which tells NHibernate which database we are using. Add a new file to the project called hibernate.cfg.xml with the following contents:

```
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-configuration xmlns="urn:nhibernate-configuration-2.2">
    <session-factory>
      <property name="connection.provider">NHibernate.Connection.
DriverConnectionProvider</property>
      <property name="dialect">NHibernate.Dialect.
Oracle10gDialect</property>
      <property name="connection.driver_
class">NHibernateOracleExample.Driver.OracleDriver,
NHibernateOracleExample</property>
      <property name="connection.connection_string">Host=oraserve
r;Password=tiger;Port=1521;SID=orcl;User ID=scott</property>
      <property name="show_sql">true</property>
      <property name="proxyfactory.factory_class">NHibernate.
ByteCode.Castle.ProxyFactoryFactory, NHibernate.ByteCode.
Castle</property>
    </session-factory>
</hibernate-configuration>
```

This file tells NHibernate the type of connection we are using, the type DBMS we are connecting to, as well as which IDriver implementation we are using to connect to the database. Make sure to set the Copy to Output Directory property to Copy Always.

Now that we've added the configuration file we can run our application and verify that it creates our schema and persists our object without error. NHibernate automatically generates the DDL required as well as the ADO.NET objects needed to execute the SQL.

```
product.Name = "Cherry";
  repository.Update(product);
  product.Name = "Apple";
  product = repository.GetById(product.Id); // product.Name is
back to "Cherry"
```

CONCLUSION

NHibernate allows relatively quick and simple use of object relational mapping with minimal intrusion into the classes that are to be persisted; no special base or wrapper classes are needed. Use of the 100% managed code providers not only greatly simplifies your NHibernate deployments but also accelerates the performance and scalability, and enhance the security of them as well. In choosing Progress DataDirect Connect *for* ADO.NET providers for Oracle, Sybase, DB2, and SQL Server, you not only get connected quicker, but have a smaller yet faster NHibernate application runtime.

ADDITIONAL RESOURCES

- nhforge.org/
- web.datadirect.com/products/net/index.html
- web.datadirect.com/products/net/managed-code.html

PROGRESS