Remodeling .Net Pet Shop

with

NJDX™
The KISS OR-Mapper™ for .NET

A Project Report

By
Prajakta Bahekar
Renu Kolli
Neelima Sangeneni

February 7, 2006

ABSTRACT

The .NET Pet Shop application is designed to show the .NET best practices for building enterprise n-tier applications. In this ASP.NET based web application, classes are defined to represent domain model objects like products, users, addresses, and orders. These domain model objects are persisted in relational databases (e.g., SQL Server, Oracle) using a data access layer (DAL). The current DAL implementation, which uses raw SQL and low-level database APIs, is pretty hard to understand and maintain. This report describes how such a data access layer can be replaced with a much simpler and shorter (35% less lines of code) implementation using NJDX OR-Mapping technology from Software Tree. In addition to greatly simplifying the architecture, the NJDX approach provides greater flexibility and delivers superb performance.

Software Tree
Simplify Data Integration

Table of Contents

Introduction

The .NET Pet Shop application is now in its third revision and is designed to show the .NET best practices for building enterprise n-tier applications, which may need to support a variety of database platforms and deployment models. This report documents how we remodeled the current Pet Shop application using Software Tree's NJDX Object-Relational Mapper (OR-Mapper) product and outlines the advantages of this approach.

Figure 1. .NET Pet Shop Welcome Page

The original application consists of a Web tier built with ASP.NET Web Forms which uses "code-behind" to separate the application HTML from the user interface code. Classes are defined to represent domain model objects like products, users, addresses, and orders. The middle tier contains business components to control the application logic, which communicates with a relational database through a data access layer (DAL) to persist domain model objects. The current DAL implementation uses raw SQL and low-level database APIs to perform the data exchange for the domain model objects with relational databases. This implementation is difficult to understand, modify, and maintain. For more details on the Pet Shop application, please check here.

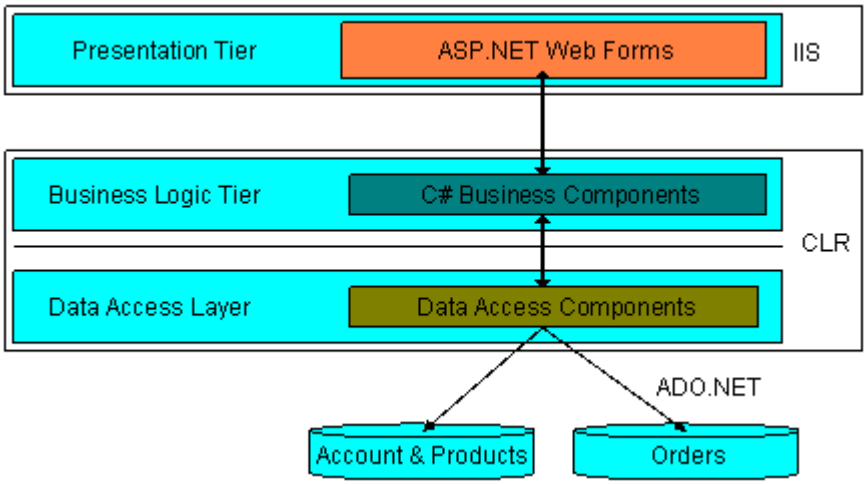


Figure 2. .NET Pet Shop high-level logical architecture

NJDX, the KISS OR-Mapper™ for .NET, seamlessly bridges the gap between the .NET object model and SQL relational model. NJDX employs a clean, dynamic, and meta-data driven programming methodology supporting pure domain object models, resulting in applications that are simpler and faster to develop, and easier to modify and maintain. For more details on NJDX, click here.

We were put to the challenge of hosting the original Pet Shop 3.0 application over NJDX OR-Mapper without making any changes to the existing object model, database schema, or client code. Essentially our job was to create a new NJDX-based data access layer (DAL) that would cleanly replace the old (SQLServer) DAL layer without affecting any other parts of the program or the database.

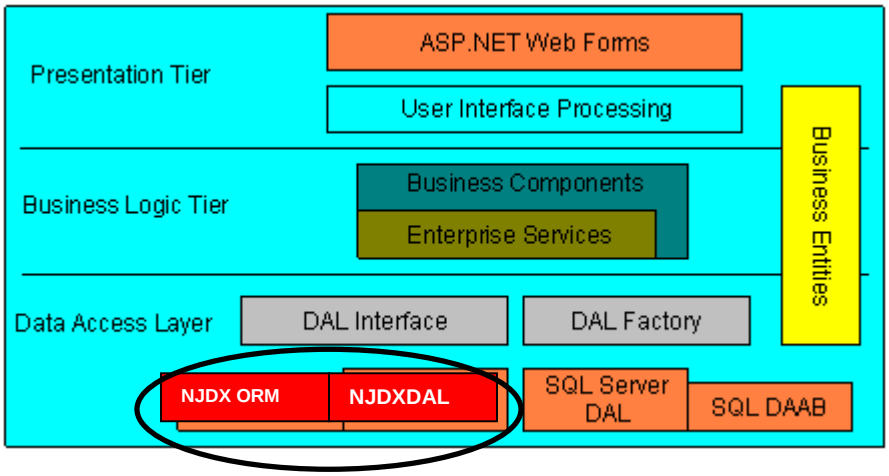


Figure 3. .NET Pet Shop 3.0 application architecture

Our project involved creating a DAL (NJDXDAL) that uses NJDX OR-Mapper for data access

After analyzing the current implementation and the requirements, we divided our project in two parts:

1- Mapping Design
2- DAL Implementation using NJDX

Before describing the details of the implementation, we will outline some basic NJDX OR-Mapper concepts. Then we will present an overview of the Mapping Design, and an outline of a DAL implementation using NJDX.

The report concludes with the advantages of the new NJDX-based architecture and an appendix that shows an example of the dramatic reduction in the size and complexity of the data access code.

NJDX OR-Mapper Concepts

Here we describe a few concepts of the NJDX OR-Mapper to help understand our project and this report better.

Object-Relational Mapping File (ORMFile)

An object-relational mapping (OR-Mapping) specification contains mapping information for all the persistent classes belonging to an application domain. The specification includes, among other things, table names, primary key attributes and object relationships. NJDX provides an innovative declarative way of specifying human-readable and user-friendly object-relational mapping information based on a simple grammar. A text file (ORMFile) containing this mapping specification can be generated using a text editor, NJDXStudio, a modeling tool, or even programmatically.

A mapping specification can correspond to only one database. While a mapping specification cannot span multiple databases, multiple mapping specifications can be defined for the same database.

Mapping Unit

A mapping specification (identified by an ORMFile) defines a mapping unit such that all interactions corresponding to the mapped classes and sequences in that specification share the same transaction manager and pool of database connections. Multiple threads of an application may share a mapping unit. An application can work with multiple mapping units.

Domain Model Assembly (DM_ASSEMBLY)

A domain model assembly is a unit of compiled domain model classes that are persisted using NJDX. That is, the mapping unit classes whose mappings are defined in the mapping specification (e.g., *petshop.jdx* file) are compiled and assembled in a domain model assembly (e.g., DomainModel.dll). This domain model assembly is identified by DM_ASSEMBLY parameter (e.g., DM_ASSEMBLY=*'<Path>/PetShop.Model.dll'*) in the Database URL or JDXURL.

JXResource

A JXResource provides the facilities to work with a mapping unit. A JXResource contains handles for JXSession and JDXS objects. JXSession provides transactional methods (like tx_begin and tx_commit) and JDXS provides, among other things, methods (like query, insert, update, and delete) to store and retrieve domain model objects.

JXResourcePool

A JXResourcePool provides a pool of JXResource(s) for a particular mapping unit based on an OR-Mapping specification. A JXResourcePool allows you to create multiple JXResource components in an extensible pool and provides you with methods for thread-safe sharing of these components.

For each object-relational mapping file, there is a corresponding domain model assembly, mapping unit, and JXResourcePool.

Mapping Design

The **Model** project of the original Pet Shop application contains domain model classes for holding data and transporting it to different tiers of the application. The data access layer (DAL) interacts with the underlying database to exchange data with these domain model objects.

Unfortunately, the relational data model was not properly normalized, and did not correlate well with the original object model. The existing implementation bridged discrepancies between the object and data models by using numerous complex SQL statements incorporating complicated joins buried inside the DAL code. Figuring out the existing mapping to satisfy the application requirements was one of the biggest challenges.

This is what we did as part of the Mapping Design project:

- Created some new domain model classes to facilitate clean mapping and simpler data access logic.

- Created some new views on the existing tables to simplify the mapping of domain model classes over the relational schema.

- Had to make two unavoidable changes to the existing domain model classes:
  - Added an empty no-argument constructor to the ItemInfo class.
  - Added two fields (Courier, Locale) to the OrderInfo class corresponding to two non-NULLABLE columns in the Orders table.

- Defined object relational mapping specifications for all the domain model classes based on existing tables and the newly created views. The original Pet Shop application deals with two databases (Petshop, PetshopOrders). Since, as part of the remodeling requirements, we did not want to make any changes to the existing database design, we have defined two mapping specification files (*`petshop.jdx`* and *`petshop_orders.jdx.`*)

- Some of the special specifications within the *`petshop.jdx`* mapping file include:
  - Defined a READONLY cache for the **CategoryInfo** class to speed up queries for its instances.
  - Defined a named query **productsByCategory** for the **ProductInfo** class to speed up category-based queries for its instances.

- Some of the special specifications within the *`petshop_orders.jdx`* mapping file include:
  - Defined a Sequence generator (**ORDERID_SEQ**) to easily create persistently unique OrderId for new **OrderInfo** objects. The existing implementation used an IDENTITY column for this purpose. But assigning the same OrderId to the corresponding **LineItemInfo** objects required a separate query. The new implementation simplifies the programming logic, improves performance, and creates a database independent implementation.
  - Defined an IMPLICIT_ATTRIB **OrderId** for the class **LineItemInfo**, which automatically gets initialized by NJDX based on the OrderId of the containing **OrderInfo** object.

- We utilized NJDXDemo GUI tool to easily modify and validate our mapping specifications with existing schema and data.

- Externalizing the object-relational mapping in a simple text file (no XML!) made it very easy to understand and comprehend the different entities that were involved in the data exchange and how they were related This also simplified the subsequent implementation of the DAL layer as there was absolutely no need to hard code any SQL statements in the data access logic.

DAL Implementation using NJDX

Once we figured that out and externalized the object-relational mapping specification in NJDX mapping files, the rest of the project became easy. Now we describe some of the artifacts we created and used to implement the remodeling of the .NET Pet Shop.

OR-Mapping files (petshop.jdx and petshop_orders.jdx)

The original Pet Shop application uses 2 databases (Petshop, PetshopOrders). Since we did not want to make any changes to the existing object model and the database design, we have created 2 mapping specification files (*petshop.jdx* and *petshop_orders.jdx*).

Utility classes (NJDXHandlers, Petshop_NJDXHandlers and Petshop_Orders_NJDXHandlers)

Each method in the data access layer (DAL) module needs to work with the NJDX OR-Mapping subsystem to execute transactional operations and data exchange for domain model objects. To facilitate that, we created a base utility class – **NJDXHandlers** that contains the variables and the logic (using a Template Method design pattern) for checking out and checking in a JXResource from a JXResourcePool corresponding to the underlying mapping specification. It also has utility methods to use NJDX Sequences conveniently and efficiently.

Petshop_NJDXHandlers and **Petshop_Orders_NJDXHandlers,** subclasses of NJDXHandlers, provide the actual JXResourcePools corresponding to the mapping unit described by the files *petshop.jdx* and *petshop_orders.jdx* respectively. Each of these two classes has a static JXResourcePool variable that is initialized during the application startup time in the method **Application_Start()** of the *Global.asax* module. The **Application_Start()** method first reads the mapping configuration parameters (JDXURL) from the newly introduced application variables **Petshop_JDXURL** and **Petshop_Orders_JDXURL** in the *Web.config* file.

PetShop.NJDXDAL project

To implement the Petshop DAL layer on top of NJDX OR-Mapper, we created a new dll project named **PetShop.NJDXDAL** which uses the IDAL interfaces. This project contains the above-mentioned classes **Petshop_NJDXHandlers** and **Petshop_Orders_NJDXHandlers** for JXResource handling. All the original classes of the existing (SQL Server) DAL layers have been re-implemented using NJDX. Each such DAL class now inherits from either **Petshop_NJDXHandlers** or **Petshop_Orders_NJDXHandlers** depending upon which mapping specification it needs to use. Each method in the DAL class has the following code pattern:

- *Check out a JXResource*
- *Perform data exchange and transaction operations using JXSession and JDXS methods*
- *Check in the JXResource*

After developing and verifying the above code pattern, it was a breeze to change all the DAL modules to use NJDX.

Here is an example of how simple the implementation of the GetOrder() method has become in the NJDXDAL **Order** class

```
public OrderInfo GetOrder(int orderId) {
  //Set up a return value
  OrderInfo order = null;
  try
  {
    checkoutJXResource();
    ObjectId oid = ObjectId.createObjectId
                        ("PetShop.Model.OrderInfo;_orderId="+orderId);
    order = (OrderInfo) njdxHandle.getObjectById(oid, false, 0, null);
  }
  catch (System.Exception e)
  {
    throw e;
  }
  finally
  {
    checkinJXResource();
  }
  return order;
}
```

Figure 4. New NJDXDAL.Order.GetOrder() method – short and simple

The original SQLServer DAL implementation of the same method takes more than double the number of lines of code which are embedded with complicated SQL statements and verbose processing logic related to SqlDataReader. Please see **Appendix A** for the full listings of the **Order** class corresponding to SQLServerDAL and NJDXDAL implementations.

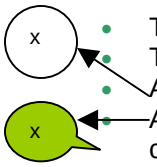
Since the new implementation does not use any explicit SQL commands, we removed the *SQLHelper* module from the new project.

Finally, under **appSettings** configuration settings in the *Web.config* file, we changed the value for the keys **WebDAL** and **OrderDAL** from **PetShop.SQLServer** to **PetShop.NJDXDAL**.

Essentially, JXResourcePools (in **Petshop_NJDXHandlers** and **Petshop_Orders_NJDXHandlers** classes) for NJDX OR-Mapping specifications are initialized (injected) at the (ASP.NET) application startup time in the method **Application_Start()** of the *Global.asax* module. Runtime data exchange between domain model objects and relational data happens through methods in **PetShop.NJDXDAL** modules, which in turn use JXResource(s) from the pre-initialized JXResourcePool(s).

Rebuilding the Pet Shop application after the above changes made the NJDX DAL code operational.

Here is a high-level process flow diagram. To avoid clutter, we are not showing the initialization of **Petshop_Orders_NJDXHandlers** (corresponding to *`petshop_orders.jdx`*).



- The yellow blocks show web and business logic tiers.
- The teal blocks show the DAL tier utilizing NJDX for data access.
- A round circle with a number represents a high-level action.
- A call out circle (in green color) with a number shows the direction of data transfer or control transfer corresponding to the high-level action with the same number.

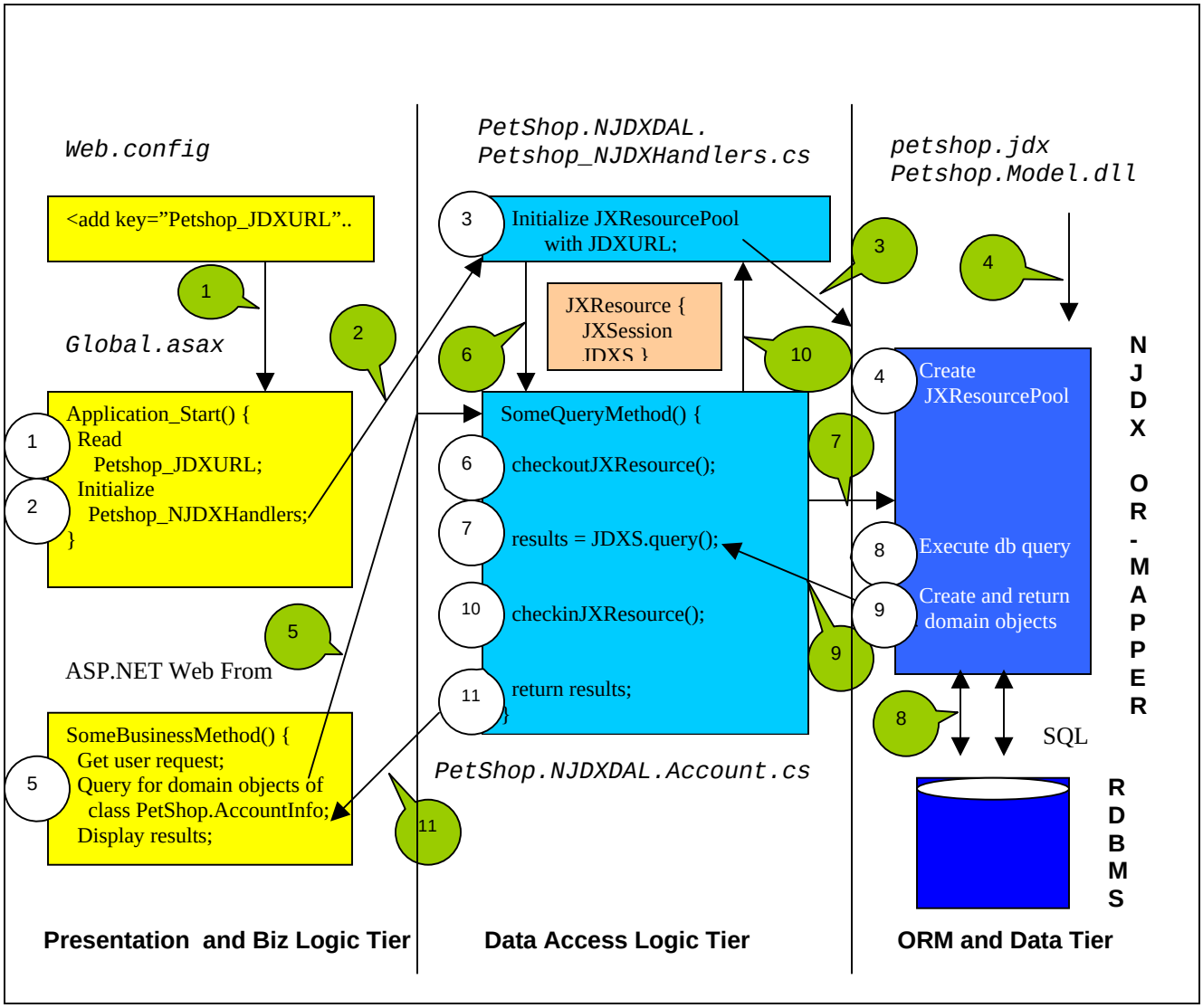


Fig 5: A Process Flow Diagram showing initializations and usage of NJDX DAL layer in the .NET Pet Shop Application

- Actions 1 to 4 show initialization of the NJDX DAL layer
- Actions 5 to 11 show the process flow in response to a query request

Summary

By eliminating the complex spaghetti code involving tedious SQL statements and their elaborate processing, we have achieved a cleaner design, a smaller and more intuitive code base, and an apparently higher performance implementation of the .NET Pet Shop application using Software Tree's NJDX OR-Mapper. For example, as **Appendix A** shows, the new implementation with NJDX shrunk the code size of the Order module from 234 lines in SQLServerDAL implementation to 68 lines in NJDXDAL implementation (a 70% reduction)! Overall, the NJDXDAL layer has 35% less lines of code (LOC) compared to the SQLServer DAL implementation. Best of all, the new implementation can work with any backend database including SQL Server, Oracle, and IBM DB2 because NJDX provides a database agnostic OR-Mapping solution.

Achieving such impressive results under strict constraints of only making changes to the DAL layer of a sophisticated ASP.NET application, and in a short timeframe of a few weeks, is a great demonstration of NJDX's power, flexibility, and ease-of-use.

It was quite a challenging project, but NJDX OR-Mapper made it painless and enjoyable. This project provides a detailed example of how an efficient data access layer of an enterprise class ASP.NET application can easily be developed using NJDX. It also demonstrates some best practice examples of using NJDX APIs.

The remodeled Pet Shop application ships with the NJDX OR-Mapper software. Please visit Software Tree's web (http://www.softwaretree.com) for more details.

Acknowledgements

We want to thank Damodar Periwal, the architect of NJDX, for his guidance and help throughout the project.

We would like to thank Julian Keith Loren, MCSD, MCAD, for reviewing this report and offering valuable feedback to improve the contents of the material.

Figures 1, 2, and 3 have been taken and adapted from the first referenced paper **Microsoft .NET Pet Shop 3.x: Design Patterns and Architecture of the .NET Pet Shop**.

References

Microsoft .NET Pet Shop 3.x: Design Patterns and Architecture of the .NET Pet Shop by Leake, Gregory and Duff James.
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/petshop3x.asp
(The above reference also has a link to download PetShop 3.0 installer)

Software Tree's web site
http://www.softwaretree.com
(The above reference also has a link to download the NJDX OR-Mapper product)

The KISS Principles for ORM – A White Paper
http://www.softwaretree.com/products/njdx/whitepaper/KISSPrinciples.pdf

Appendix A

In this appendix, we provide the program listings of the Order.cs module corresponding to SQLServerDAL and NJDXDAL implementations.

PetShop.SQLServerDAL.Order.cs (234 lines of program code)

```
using System;
using System.Data;
using System.Diagnostics;
using System.Collections;
using System.Data.SqlClient;
using PetShop.Model;
using PetShop.IDAL;

namespace PetShop.SQLServerDAL {

  public class Order : IOrder{

        //Static constants
        private const string SQL_INSERT_ORDER = "Declare @ID int; Declare @ERR int; INSERT
INTO Orders VALUES(@UserId, @Date, @ShipAddress1, @ShipAddress2, @ShipCity, @ShipState,
@ShipZip, @ShipCountry, @BillAddress1, @BillAddress2, @BillCity, @BillState, @BillZip,
@BillCountry, 'UPS', @Total, @BillFirstName, @BillLastName, @ShipFirstName,
@ShipLastName, @CardNumber, @CardExpiration, @CardType, 'US_en'); SELECT @ID=@@IDENTITY;
INSERT INTO OrderStatus VALUES(@ID, @ID, GetDate(), 'P'); SELECT @ERR=@@ERROR;";
        private const string SQL_INSERT_ITEM = "INSERT INTO LineItem VALUES( ";
        private const string SQL_SELECT_ORDER = "SELECT o.OrderDate, o.UserId, o.CardType,
o.CreditCard, o.ExprDate, o.BillToFirstName, o.BillToLastName, o.BillAddr1, o.BillAddr2,
o.BillCity, o.BillState, BillZip, o.BillCountry, o.ShipToFirstName, o.ShipToLastName,
o.ShipAddr1, o.ShipAddr2, o.ShipCity, o.ShipState, o.ShipZip, o.ShipCountry,
o.TotalPrice, l.ItemId, l.LineNum, l.Quantity, l.UnitPrice FROM Orders as o, lineitem as
l WHERE o.OrderId = @OrderId AND o.orderid = l.orderid";
        private const string PARM_USER_ID = "@UserId";
        private const string PARM_DATE = "@Date";
        private const string PARM_SHIP_ADDRESS1 = "@ShipAddress1";
        private const string PARM_SHIP_ADDRESS2 = "@ShipAddress2";
        private const string PARM_SHIP_CITY = "@ShipCity";
        private const string PARM_SHIP_STATE = "@ShipState";
        private const string PARM_SHIP_ZIP = "@ShipZip";
        private const string PARM_SHIP_COUNTRY = "@ShipCountry";
        private const string PARM_BILL_ADDRESS1 = "@BillAddress1";
        private const string PARM_BILL_ADDRESS2 = "@BillAddress2";
        private const string PARM_BILL_CITY = "@BillCity";
        private const string PARM_BILL_STATE = "@BillState";
        private const string PARM_BILL_ZIP = "@BillZip";
        private const string PARM_BILL_COUNTRY = "@BillCountry";
        private const string PARM_TOTAL = "@Total";
        private const string PARM_BILL_FIRST_NAME = "@BillFirstName";
        private const string PARM_BILL_LAST_NAME = "@BillLastName";
        private const string PARM_SHIP_FIRST_NAME = "@ShipFirstName";
        private const string PARM_SHIP_LAST_NAME = "@ShipLastName";
        private const string PARM_CARD_NUMBER = "@CardNumber";
        private const string PARM_CARD_EXPIRATION = "@CardExpiration";
        private const string PARM_CARD_TYPE = "@CardType";
        private const string PARM_ORDER_ID = "@OrderId";
        private const string PARM_LINE_NUMBER = "@LineNumber";
        private const string PARM_ITEM_ID = "@ItemId";
        private const string PARM_QUANTITY = "@Quantity";
        private const string PARM_PRICE = "@Price";

        public int Insert(OrderInfo order) {

          int orderId = 0;
          String strSQL = null;
          try {

                // Get each commands parameter arrays
```

```
SqlParameter[] orderParms = GetOrderParameters();
SqlParameter statusParm = new SqlParameter
                              (PARM_ORDER_ID, SqlDbType.Int);
SqlCommand cmd = new SqlCommand();
// Set up the parameters
orderParms[0].Value = order.UserId;
orderParms[1].Value = order.Date;
orderParms[2].Value = order.ShippingAddress.Address1;
orderParms[3].Value = order.ShippingAddress.Address2;
orderParms[4].Value = order.ShippingAddress.City;
orderParms[5].Value = order.ShippingAddress.State;
orderParms[6].Value = order.ShippingAddress.Zip;
orderParms[7].Value = order.ShippingAddress.Country;
orderParms[8].Value = order.BillingAddress.Address1;
orderParms[9].Value = order.BillingAddress.Address2;
orderParms[10].Value = order.BillingAddress.City;
orderParms[11].Value = order.BillingAddress.State;
orderParms[12].Value = order.BillingAddress.Zip;
orderParms[13].Value = order.BillingAddress.Country;
orderParms[14].Value = order.OrderTotal;
orderParms[15].Value = order.BillingAddress.FirstName;
orderParms[16].Value = order.BillingAddress.LastName;
orderParms[17].Value = order.ShippingAddress.FirstName;
orderParms[18].Value = order.ShippingAddress.LastName;
orderParms[19].Value = order.CreditCard.CardNumber;
orderParms[20].Value = order.CreditCard.CardExpiration;
orderParms[21].Value = order.CreditCard.CardType;
foreach (SqlParameter parm in orderParms)
                      cmd.Parameters.Add(parm);

// Create the connection to the database
using (SqlConnection conn = new
       SqlConnection(SQLHelper.CONN_STRING_DTC_ORDERS)) {

       // Open the database connection
       // Insert the order status
       strSQL = SQL_INSERT_ORDER;
       SqlParameter[] itemParms ;
       // For each line item, insert an orderline record
       int i = 0;
       foreach (LineItemInfo item in order.LineItems) {
              strSQL = strSQL + SQL_INSERT_ITEM + " @ID" + ",
              @LineNumber"+i + ", @ItemId" + i+ ", @Quantity"
              + i + ", @Price" + i + "); SELECT
              @ERR=@ERR+@@ERROR;";

              //Get the cached parameters
              itemParms = GetItemParameters(i);

              itemParms[0].Value = item.Line;
              itemParms[1].Value = item.ItemId;
              itemParms[2].Value = item.Quantity;
              itemParms[3].Value = item.Price;
              //Bind each parameter
              foreach (SqlParameter parm in itemParms)
                     cmd.Parameters.Add(parm);
                     i++;
       }

       conn.Open();
       cmd.Connection = conn;
       cmd.CommandType = CommandType.Text;
       cmd.CommandText = strSQL + "SELECT @ID, @ERR";

       // Read the output of the query, should return
       // orderid and error count
       using (SqlDataReader rdr = cmd.ExecuteReader
                      (CommandBehavior.CloseConnection)){

              //Read the result
              rdr.Read();
```

```
                        // If the error count is not zero
                        // throw an exception
                        if (rdr.GetInt32(1) != 0)
                                throw new Exception("DATA INTEGRITY
                                                ERROR ON ORDER INSERT –
                                                ROLLBACK ISSUED");
                        //Fetch the orderId
                        orderId = rdr.GetInt32(0);
                }
                //Clear the parameters
                cmd.Parameters.Clear();
        }
  }catch(Exception e){
        throw e;
  }finally{
  }
  return orderId;
}

/// <summary>
/// Read an order from the database
/// </summary>
/// <param name="orderId"></param>
/// <returns></returns>
public OrderInfo GetOrder(int orderId) {

  //Create a parameter
  SqlParameter parm = new SqlParameter(PARM_ORDER_ID,
                                SqlDbType.Int);
  parm.Value = orderId;

  //Execute a query to read the order
  using (SqlDataReader rdr =
        SQLHelper.ExecuteReader(SQLHelper.CONN_STRING_DTC_ORDERS,
         CommandType.Text, SQL_SELECT_ORDER, parm)) {

        if (rdr.Read()) {

                //Generate an order header from the first row
                CreditCardInfo creditCard = new CreditCardInfo
                        (rdr.GetString(2), rdr.GetString(3),
                        rdr.GetString(4));
                AddressInfo billingAddress = new AddressInfo
                        (rdr.GetString(5), rdr.GetString(6),
                        rdr.GetString(7), rdr.GetString(8),
                        rdr.GetString(9), rdr.GetString(10),
                        rdr.GetString(11), rdr.GetString(12), null);
                AddressInfo shippingAddress = new AddressInfo
                        (rdr.GetString(13), rdr.GetString(14),
                         rdr.GetString(15), rdr.GetString(16),
                         rdr.GetString(17), rdr.GetString(18),
                         rdr.GetString(19), rdr.GetString(20), null);

                OrderInfo order = new OrderInfo(orderId,
                                        rdr.GetDateTime(0), rdr.GetString(1),
                                        creditCard, billingAddress,
                                        shippingAddress, rdr.GetDecimal(21));


                ArrayList lineItems = new ArrayList();
                LineItemInfo item = null;

                //Create the lineitems from the first row and
                // subsequent rows
                do{
                        item = new LineItemInfo(rdr.GetString(22),
                                        string.Empty, rdr.GetInt32(23),
                                        rdr.GetInt32(24), rdr.GetDecimal(25));

                        lineItems.Add(item);
                }while(rdr.Read());
```

```
                        order.LineItems = (LineItemInfo[])
                                lineItems.ToArray(typeof(LineItemInfo));

                        return order;
                }
        }
        return null;
                }

    /// <summary>
    /// Internal function to get cached parameters
    /// </summary>
    /// <returns></returns>
    private static SqlParameter[] GetOrderParameters() {
        SqlParameter[] parms =
                        SQLHelper.GetCachedParameters(SQL_INSERT_ORDER);

        if (parms == null) {
          parms = new SqlParameter[] {
                new SqlParameter(PARM_USER_ID, SqlDbType.VarChar, 80),
                new SqlParameter(PARM_DATE, SqlDbType.DateTime, 12),
                new SqlParameter(PARM_SHIP_ADDRESS1, SqlDbType.VarChar, 80),
                new SqlParameter(PARM_SHIP_ADDRESS2, SqlDbType.VarChar, 80),
                new SqlParameter(PARM_SHIP_CITY, SqlDbType.VarChar, 80),
                new SqlParameter(PARM_SHIP_STATE, SqlDbType.VarChar, 80),
                new SqlParameter(PARM_SHIP_ZIP, SqlDbType.VarChar, 50),
                new SqlParameter(PARM_SHIP_COUNTRY, SqlDbType.VarChar, 50),
                new SqlParameter(PARM_BILL_ADDRESS1, SqlDbType.VarChar, 80),
                new SqlParameter(PARM_BILL_ADDRESS2, SqlDbType.VarChar, 80),
                new SqlParameter(PARM_BILL_CITY, SqlDbType.VarChar, 80),
                new SqlParameter(PARM_BILL_STATE, SqlDbType.VarChar, 80),
                new SqlParameter(PARM_BILL_ZIP, SqlDbType.VarChar, 50),
                new SqlParameter(PARM_BILL_COUNTRY, SqlDbType.VarChar, 50),
                new SqlParameter(PARM_TOTAL, SqlDbType.Decimal, 8),
                new SqlParameter(PARM_BILL_FIRST_NAME, SqlDbType.VarChar, 80),
                new SqlParameter(PARM_BILL_LAST_NAME, SqlDbType.VarChar, 80),
                new SqlParameter(PARM_SHIP_FIRST_NAME, SqlDbType.VarChar, 80),
                new SqlParameter(PARM_SHIP_LAST_NAME, SqlDbType.VarChar, 80),
                new SqlParameter(PARM_CARD_NUMBER, SqlDbType.VarChar, 80),
                new SqlParameter(PARM_CARD_EXPIRATION, SqlDbType.Char, 10),
                new SqlParameter(PARM_CARD_TYPE, SqlDbType.VarChar, 80)};

           SQLHelper.CacheParameters(SQL_INSERT_ORDER, parms);
        }
    return parms;
    }

    private static SqlParameter[] GetItemParameters(int i) {
        SqlParameter[] parms = SQLHelper.GetCachedParameters(SQL_INSERT_ITEM
                                                        +i);

        if (parms == null) {
                parms = new SqlParameter[] {
                        //new SqlParameter(PARM_ORDER_ID + i, SqlDbType.Int, 4),
                        new SqlParameter(PARM_LINE_NUMBER + i, SqlDbType.Int, 4),
                        new SqlParameter(PARM_ITEM_ID+i, SqlDbType.Char, 10),
                        new SqlParameter(PARM_QUANTITY+i, SqlDbType.Int, 4),
                        new SqlParameter(PARM_PRICE+i, SqlDbType.Decimal, 8)};

                SQLHelper.CacheParameters(SQL_INSERT_ITEM+i, parms);
        }
        return parms;
        }
      }
} // End of PetShop.SQLServerDAL.Order.cs
```

PetShop.NJDXDAL.Order.cs (68 lines of program code)

```
using System;
using PetShop.Model;
using PetShop.IDAL;
using com.softwaretree.jx;
using com.softwaretree.jdx;

namespace PetShop.NJDXDAL {

        public class Order : Petshop_Orders_NJDXHandlers, IOrder{

          public int Insert(OrderInfo order) {

                int orderId = 0;
                try
                {
                   checkoutJXResource();

                   orderId = (int) getNextOrderId();
                   order.OrderId = orderId;

                   jxSession.tx_begin();
                   // The folowing statement is a work-around for inserting
                   // a value in an IDENTITY column (OrderId)
                   njdxHandle.SQLStatement("SET identity_insert Orders ON",0, null);

                   // insert the order object. Default is DEEP insert,
                   // so lineItems will also be inserted.
                   njdxHandle.insert(order, 0, null);
                   jxSession.tx_commit();
                }
                catch (System.Exception e)
                {
                   jxSession.tx_rollback();
                   throw e;
                }
                finally
                {
                   checkinJXResource();
                }
                   return orderId;
           }

           /// <summary>
           /// Read an order from the database
           /// </summary>
           /// <param name="orderId"></param>
           /// <returns></returns>
           public OrderInfo GetOrder(int orderId) {

                //Set up a return value
                OrderInfo order = null;
                try
                {
                   checkoutJXResource();
                   ObjectId oid = ObjectId.createObjectId
                               ("PetShop.Model.OrderInfo;_orderId="+orderId);
                   order = (OrderInfo)njdxHandle.getObjectById(oid, false, 0, null);
                }
                catch (System.Exception e) {
                   throw e;
                }
                finally {
                   checkinJXResource();
                }
                   return order;
                }
        }
} // End of PetShop.NJDXDAL.Order.cs
```

Trademarks

Software Tree, Software Tree logo, JDX, NJDX, NJDX logo, and 'The KISS OR-Mapper' are trademarks of Software Tree, Inc. Windows, .NET, Visual Studio .NET, and C#.NET are registered trademarks of Microsoft Corporation.