

HOME

BLOGS

ARTICLES

FAQS

SNIPPETS

ABOUT

CONTACT

LOGIN

Classic ASP ASP.NET 1.x ASP.NET 2.0 Dreamweaver (MX / MX 2004 / 8) Web General .NET General .NET Server Controls .NET 2.0 General .NET CF Security

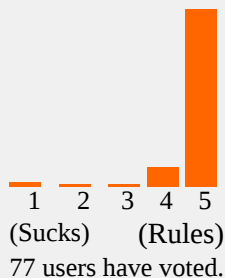
Details

**QuickDocId** 419**Written by** Imar Spaanjaars**Posted** 02/17/2007 17:29**Modified** 04/25/2007 00:03**Page views** 18613**Listened to** Hits of Sunshine (For Allen Ginsberg) by Sonic Youth (Track 7 from the album: A Thousand Leaves)[Print this Page](#)

Rate this item



Like this article? Or do you think it sucks? Make yourself heard by casting your vote below.



1 2 3 4 5
(Sucks) (Rules)

Building Layered Web Applications with Microsoft ASP.NET 2.0 - Part 2

Update!! 04-25-2007 - There is now also a VB.NET version of the code available for download. You find the download at the end of this article. For more information about the translation, check out [this blog post](#).

Update!! 04-24-2007 - Fixed a bug in the stored procedure that deletes a Contactperson, based on feedback from reader csharpdev. This also effects the downloadable code for this article, which now also has the update applied.

Update!! 02-25-2007 - I made some changes to the way the lists with contact data are used. Previously, the ContactPerson implemented properties of type List<T> directly, but I have now created separate types for the lists.

This is part two of the article series "Building Layered Web Applications" that shows you how to build N-Layer applications with Microsoft ASP.NET 2.0. These articles teach you how to design, build and use custom business objects in your web application. The target audience for this series are developers that are ready to make the switch from using SqlDataSource controls to ObjectDataSource controls with custom business objects. Experience with ASP.NET 2 and C# is necessary while some knowledge about object oriented design certainly helps. The design I am going to show you in these articles is a simplified version of a design you would use in a real world application. It doesn't feature all the necessary functionality your application needs, but instead focuses on the underlying concepts.

Part one dealt with the design of the application: what business objects do you need to fulfill the requirements of the application. What should these objects be capable of and how do they look. This article (part two) continues where the previous article stopped: it shows you how to code the classes that were designed in part one. You'll see how to implement the data access methods and database code and you'll learn how you can connect these classes. **Part three** then deals with using the business objects in a web application. You'll see how to make use of the out-of-the-box ASP.NET 2.0 controls together with the business objects coded in this part.

If you haven't read part one yet, you should really read it first, as this article uses many concepts that have been explained earlier. Afterwards, check out part three where you'll see the concepts from this article in an actual web site. The entire series (including this current article) can be found here:

- [Building Layered Web Applications with Microsoft ASP.NET 2.0 - Part 1](#)
- [Building Layered Web Applications with Microsoft ASP.NET 2.0 - Part 2](#)
- [Building Layered Web Applications with Microsoft ASP.NET 2.0 - Part 3](#)

The article uses a SQL Server 2005 Express database which is easy to use in development scenario's. However, the downloads for this series also come with the T-SQL scripts to recreate the database in SQL Server 2000 or SQL Server 2005. You'll find the download link at the end of this article. Besides the forementioned SQL scripts and database, the download also contains the full source for the demo application in C#.

Introduction

In part 1 you saw the design of the classes that make up the Contact Manager application. You saw the classes in the Business Objects layer, the Business Logic Layer and you saw a number of classes in the DAL (Data Access Layer).

In this installment, I'll show you the code for these classes. You'll see how to implement the public properties for each of the classes in the Business Objects layer, how to write the code for methods like `GetItem` and `GetList`, and you'll see how to connect to the database in a safe, transactional manner for inserts and updates. But, before we dig into the code, let's take a brief look at the class design again. With the class design in mind, it's easier to make decisions for the layout of your project. Although you could simply drop all your classes in the `App_Code` folder or in a separate class library project all under the same namespace, it's often easier to put them in separate folders and namespaces so the code is easier to find. The following figure shows the organization of the classes in the application:

Classes and Enums in the `Spaanjaars.ContactManager.BO` Namespace

ContactPerson
Class

ContactPersonList
Class

→ `List<ContactPerson>`

Address
Class

AddressList
Class

→ `List<Address>`

EmailAddress
Class

EmailAddressList
Class

→ `List<EmailAddress>`

PhoneNumber
Class

PhoneNumberList
Class

→ `List<PhoneNumber>`

PersonType
Enum

ContactType
Enum

Classes in the `Spaanjaars.ContactManager.Bll` Namespace

ContactPersonManager
Class

AddressManager
Class

EmailAddressManager
Class

PhoneNumberManager
Class

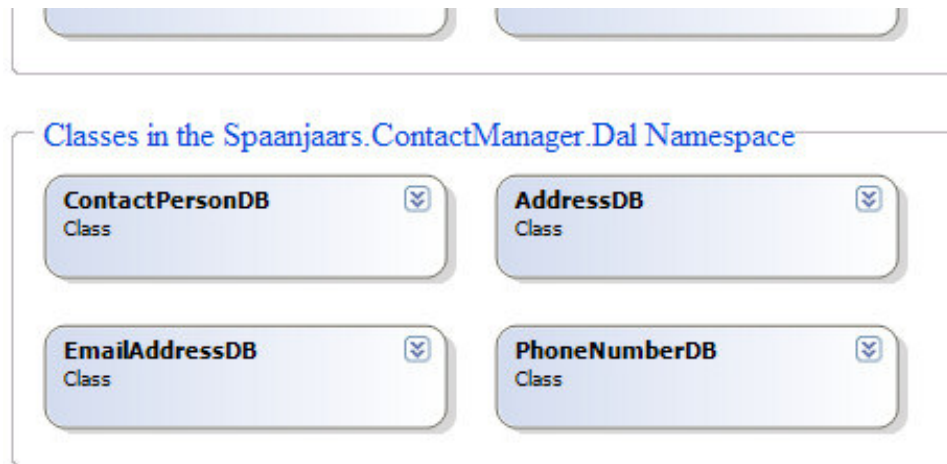


Figure 1 - The Full Class Diagram for the Contact Person Manager Application

At the top of the image you see the business objects in the BO namespace. These classes are placed in the `Spaanjaars.ContactManager.BO` namespace, following a best practice in namespace names where each namespace is named in the format *CompanyName.ProjectName.Layer*. I also put the enumerations in that layer, as they are used directly as types for certain properties of the classes in the same layer.

All the `*Manager` classes are put in the `Spaanjaars.ContactManager.BLL` namespace and similarly, I put the `*DB` classes in the `Spaanjaars.ContactManager.Dal` namespace.

To continue this separation, I also used different folders inside the `App_Code` folder of the application. In a larger application you may want to use three separate class libraries instead. The Business Layer class library would then have a reference to the Business Objects and Data Access class libraries, the DAL would also have a reference to the Business objects layer, while the final web site would reference the BO and BLL layers. In the sample application for this article, I decided to keep things simple and store everything in separate folders under `App_Code`, but the exact same principles would apply if you'd be using class library projects.

So, after I created a brand new web site in Visual Studio 2005 (I am using the Professional Edition, but you can follow along if you have the Express Edition), I added four folders under the special ASP.NET `App_Code` folder, and put a number of empty class files in them, each one named after the class or enum from the diagram in Figure 1. I ended up with the following Solution Explorer:

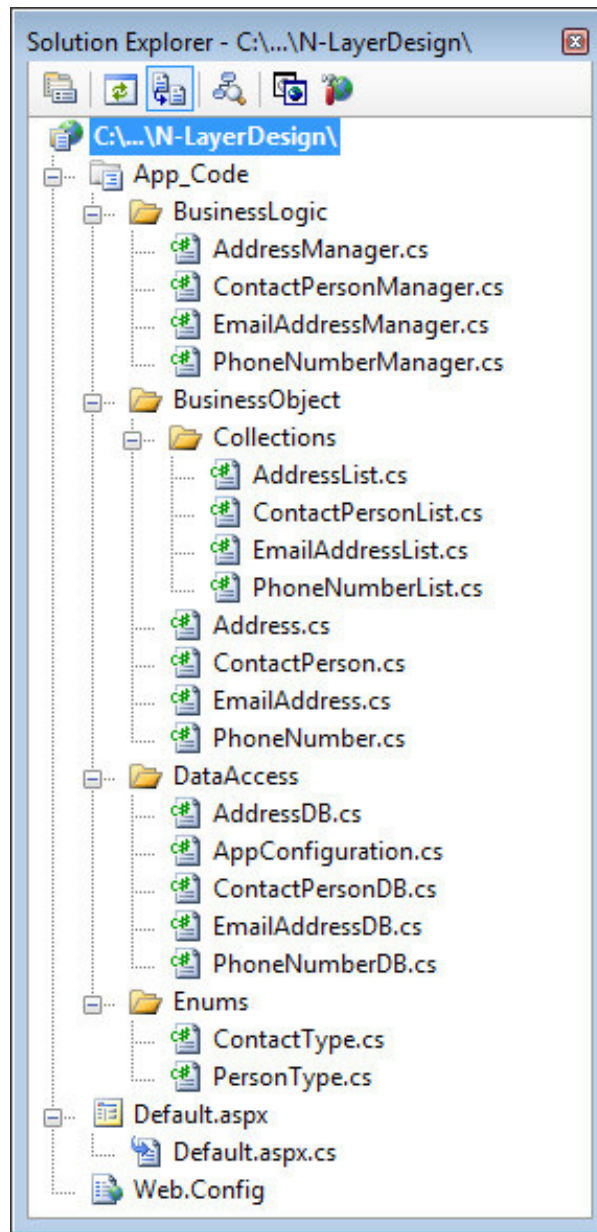


Figure 2 - The Solution Explorer for the Contact Person Manager Application

Each class in the **BusinessLogic** folder is put in the **Bll** namespace, files in the **BusinessObject** folder fall under the **BO** namespace while the classes in the **DataAccess** folder are put in the **Da1** namespace. For example, the **Address** class in the **BusinessObject** folder looks like this:

```
namespace Spaanjaars.ContactManager.BO
{
    public class Address
    {
        // Implementation here
    }
}
```

```
}
}
```

Although the enumerations actually fall under the BO namespace, I added them in a separate folder so they are easier to find.

Now that you've seen the basic structure of the site, let's take a look at the actual code for the classes.

Code Organization in the Contact Person Manager Application

Before I show you the specific implementation of the classes, let's first take a look at the basic structure of the class files. To make it easier to locate and maintain code, I often wrap them in `#region / #endregion` constructs. I typically use at least four different regions although I don't use all of them in every file. Figure 3 shows the collapsed code for a typical business object:

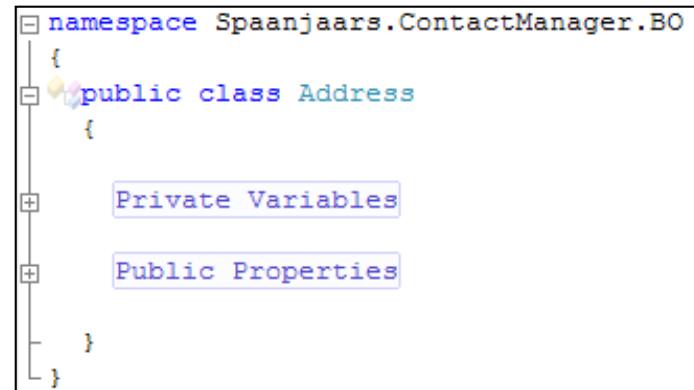


Figure 3 - The Collapsed Code for the Address Class Showing Two Code Regions

The `Private Variables` region contains all the backing variables for the public properties and optionally other private, class scoped variables for the business objects. They're often assigned a default value here as well. For example:

```
private string street = String.Empty;
```

The `Public Properties` region contains all the properties that the class exposes, like `Id`, `Street` and `HouseNumber` in the case of the `Address` class. Most of these properties use the private fields from the `Private Variables` region.

Figure 4 shows the collapsed code for a typical class in the data access layer:

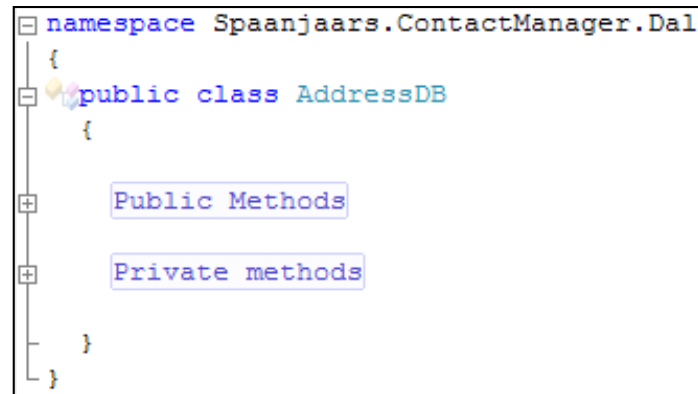


Figure 4 - The Collapsed Code for the AddressDB Class Showing Two Other Code Regions

Many classes in the sample application contain a region called `Public Methods`. This region contains the class's public methods that are used to interact with the object and the underlying database. You'll find methods like `GetItem` and `GetList` to retrieve the objects, and `Save` and `Delete` to propagate object changes back to the database. In addition, the class also contains a `Private Methods` region that contains methods that are only accessible from within the class that defines them.

Coding the Classes for the Contact Person Manager Application

With the code organization behind our back, it's time to look at the actual implementation of the classes. Since the `Address`, `EmailAddress` and `PhoneNumber` classes are very similar, I'll only look at one of them: the `EmailAddress` class. Of course, I'll also show you the `EmailAddressManager` and `EmailAddressSDB` classes as we progress through the code.

After you have seen the `EmailAddress` class, I'll take a detailed look at the `ContactPerson` class. Since this class is a bit more complex than the other three, it helps to understand the other three classes before you look at `ContactPerson`.

The EmailAddress Class - Implementation

The first region in the class, called `Private Variables` contains the following code:

```
private int id = -1;
private string email = String.Empty;
private ContactType type = ContactType.NotSet;
private int contactPersonId = -1;
```

The private field `id` is initially set to `-1`. This value is used later on to determine whether an object is entirely new (`id` is still `-1`), or that it already has an associated record in the database (`id` is greater than `0`). Normally, this is considered a magic number, where a fixed number has a special meaning that cannot be derived easily by its value. Magic numbers are usually considered a bad programming practice. You can avoid magic numbers by using `Nullable Types` (new in .NET 2) where an `int` could be `null` as well to indicate it doesn't have a value. In this case, I can live with the magic number. Legal values for the `id` and `contactPersonId` fields come from identity columns in the database. These values are always greater than zero, so it's clear from the code that `-1` means something special. If you don't like this, then check out [Nullable Types](#) on the MSDN site.

The `email` field (which contains the actual e-mail address) defaults to `String.Empty` while the `type` defaults to `ContactType.NotSet`, a custom value of the `ContactType` enumeration to indicate it hasn't been set to another value before. Think of this as a `null` value for the `ContactType` type. Finally, the `contactPersonId` is set to `-1` as well.

In the `Public Properties` region of the code, you find one property for each of these fields. All of them look similar to this:

```
public string Email
{
    get
    {
        return email;
    }
    set
    {
        email = value;
    }
}
```

You can see that each of the private fields is used as a backing variable for the public property.

Earlier I said that the classes in the BO namespace, like `EmailAddress`, are "dumb" classes. They are not capable of performing any action; all they do is contain data. To work with instances of `EmailAddress`, you need to use the methods in the `EmailAddressManager` class.

Working with Instances of EmailAddress

The `EmailAddressManager` contains a number of methods that allow you to perform CRUD operations on an `EmailAddress`: Create, Read, Update and Delete. Let's take a look at Read first.

First of all, the `EmailAddressManager` class has a method called `GetItem`. This method accepts the ID of the `EmailAddress` in the database. The implementation of this method is really simple: all it does is forward the call to the `GetItem` method in the data access layer:

```
public static EmailAddress GetItem(int id)
{
    return EmailAddressDB.GetItem(id);
}
```

Although this seems like a lot of work for nothing (why not have the calling code access the DAL method directly?), it serves a distinct purpose. Although not implemented in `GetItem` for the `EmailAddressManager` class, the code in the business layer is an ideal place for things like security. Instead of directly calling `EmailAddressDB.GetItem`, you could perform a security check first to determine whether the current user has sufficient access right to perform the call. You could, for example, modify `GetItem` so it looks like this:

```
public static EmailAddress GetItem(int id, IPrincipal currentUser)
{
    if (!currentUser.IsInRole("EmailAddressManagers"))
    {
        throw new NotSupportedException(@"You're not allowed to call
            EmailAddressManager.GetItem when you're not in the
            EmailAddressManagers role");
    }
    return EmailAddressDB.GetItem(id);
}
```

You would then call this method from an ASPX page like this:

```
EmailAddress myEmailAddress = EmailAddressManager.GetItem(10, Context.User);
```

And, instead or in addition of this, you could use this method to cache the object returned by `GetItem` for some time, so you don't have to hit the database every time you need to work with the object. Another valid activity for methods in the business layer is validation. You'll see how this works when the `Save` method is discussed.

Since these kind of requirements - security, validation, caching and so on - are often application specific, it's important to implement them in the business layer.

If you implement them in the data access layer, you can't reuse that layer in an application that has different requirements or demands.

The `GetItem` method is static which means you can call it without an object instance. With this method, calling code can get an instance of `EmailAddress` with a single line of code:

```
EmailAddress myEmailAddress = EmailAddressManager.GetItem(id);
```

where `id` is the ID of a valid `EmailAddress` in the database.

As you saw, `GetItem` in the business layer simply forwards the call to `GetItem` in the `EmailAddressDB` class. That method looks like this:

```
public static EmailAddress GetItem(int id)
{
    EmailAddress myEmailAddress = null;
    using (SqlConnection myConnection =
        new SqlConnection(AppConfiguration.ConnectionString))
    {
        SqlCommand myCommand = new SqlCommand(
            "sprocEmailAddressSelectSingleItem", myConnection);
        myCommand.CommandType = CommandType.StoredProcedure;
        myCommand.Parameters.AddWithValue("@id", id);

        myConnection.Open();
        using (SqlDataReader myReader = myCommand.ExecuteReader())
        {
            if (myReader.Read())
            {
                myEmailAddress = FillDataRecord(myReader);
            }
            myReader.Close();
        }
        myConnection.Close();
    }
    return myEmailAddress;
}
```

First, this method creates a new `SqlConnection` by passing in the connection string (retrieved from the `web.config` file through the static property `ConnectionString` on the `AppConfiguration` class). It then creates and sets up a new `SqlCommand`. The command is configured to call a stored procedure called `sprocEmailAddressSelectSingleItem` which returns a single `EmailAddress` by its ID. This ID is passed to the stored procedure by adding a parameter object to the command using `AddWithValue`. The stored procedure itself is as straight forward as it can be:

```
CREATE PROCEDURE sprocEmailAddressSelectSingleItem

@id int

AS

SELECT
    Id,
    Email,
    EmailType,
    ContactPersonId
FROM
    EmailAddress
WHERE
    Id = @id
```

This procedure simple retrieves all the fields from the `EmailAddress` table for the specified ID.

The `GetItem` method in the `EmailAddressDB` class finally opens a connection to the database and then calls `ExecuteReader` to get an open `SqlDataReader`. Each row in the `SqlDataReader` implements the generic `IDataRecord` interface that gives you strongly typed access to the values within each row. The private helper method `FillDataRecord` is then used to create and fill an instance of the `EmailAddress` class based on the data in the `IDataRecord` instance:

```
private static EmailAddress FillDataRecord(IDataRecord myDataRecord)
{
    EmailAddress myEmailAddress = new EmailAddress();
    myEmailAddress.Id =
        myDataRecord.GetInt32(myDataRecord.GetOrdinal("Id"));
    myEmailAddress.Email =
        myDataRecord.GetString(myDataRecord.GetOrdinal("Email"));
    myEmailAddress.Type = (ContactType)
        myDataRecord.GetInt32(myDataRecord.GetOrdinal("EmailType"));
    myEmailAddress.ContactPersonId =
        myDataRecord.GetInt32(myDataRecord.GetOrdinal("ContactPersonId"));
    return myEmailAddress;
}
```

This method first creates a new instance of the `EmailAddress` class and then copies each of its fields from the `IDataRecord` in the `EmailAddress`. At the end of the method, the `EmailAddress` object is returned. This helper method is used to create a single instance of each class in `GetItem`, but is also used to get lists of objects in the `GetList` methods. To make it easier to use this code as the basis for an implementation of a different database, the method accepts an `IDataRecord`, an interface that all `DataReaders`, like the `SqlDataReader` and the `OleDbDataReader` implement.

The code in this method simply copies the data from the reader into the private fields of the classes by calling the appropriate `Get *` methods. These `Get *` methods expect a zero based index of the column in the `IDataRecord`. However, using "magic numbers" in your code makes it hard to maintain and read, so I used `GetOrdinal` to "translate" the name of a column in its index. Why the `Get *` methods don't have an overload that accepts a string with the column name as well is beyond me though. This may be a nice addition for the .NET 3.5 framework.

Since all the properties of the `EmailAddress` are required columns in the database, there's no need to check for `null` values in the helper method. In cases where you do expect `null` values, you need to take extra precautions, as is the case with the `Address` class. This is code from that class's `FillDataRecord` method:

```
if (!myDataRecord.IsDBNull(myDataRecord.GetOrdinal("ZipCode")))
{
    myAddress.ZipCode =
        myDataRecord.GetString(myDataRecord.GetOrdinal("ZipCode"));
}
```

Since `ZipCode` is not a required column in the `Address` table, I have to check for `null` values using `IsDBNull`. Only when the column isn't `null`, I assign the `ZipCode` property the value from the reader. Otherwise, I leave it to its default value of `String.Empty`.

At the end of `GetItem` in the data access layer, the `EmailAddress` object is returned to `GetItem` in the business logic layer which in turn returns it to the client.

As you can see, the code in the `EmailAddressDB` class is tied to a specific database provider: SQL Server. However, with minimal changes, you can make it work with Access or Oracle. Alternatively, you can write almost completely provider agnostic code using the new database provider factory model in .NET 2. Check out my book [ASP.NET 2.0 Instant Results](#) for an example of this in the chapter The Wrox Blog.

But, however you change the code in the DAL, you don't need to change the business objects or business logic layers; as long as the DAL returns an object from the BO namespace, which is shared between the Bll and the Dal namespaces, everything will work, regardless of the database you're using.

The `GetList` method in the `EmailAddress` class in the Business Layer follows a similar approach. However, since we're potentially working with more than one e-mail address, we need some way to work with a collection of `EmailAddress` records. In .NET 2, Microsoft introduced a concept called Generics that allow you to write classes that can work with different types of other classes. In the case of the `ContactPerson`, I make use of the generic `List<T>` class. The `List<T>` works very similar to an `ArrayList` in that it allows you to easily add and remove objects to a list. However, one of the shortcomings of the `ArrayList` is the fact you can store **any** object in it. So, it's possible to accidentally store both an `EmailAddress` and an `Address` in a list. This obviously can lead to problems when you want to use the list to bind it to a `GridView` for example. The `GridView` is set up to work with a single object type and will be confused (and raise an exception) when you pass it different object types. Also, because the `GridView` isn't able to determine the actual type of the object, it also can't generate strongly typed columns for you, impacting the design time experience and your productivity.

The `List<T>` on the other hand is designed to work with objects of type `T`. You determine what `T` represents when you declare a variable of type `List<T>` as in the following example that creates a list that can only hold `EmailAddress` instances:

```
List<EmailAddress> tempList = new List<EmailAddress>();
```

With this declaration, you'll get a compile time error whenever you try to add an object to the list that is not of type `EmailAddress`.

In my initial design for this application, I used `List<T>` directly. So, my `ContactPerson` class had properties of type `List<Address>`, `List<EmailAddress>` and so on. Additionally, the `GetList` methods would work with and return types of `List<T>` as well. However, after a discussion with someone I work with, I decided to create four separate **List* classes, each inheriting from its generics counterpart. For example, the `EmailAddressList` class looks like this:

```
public class EmailAddressList : List<EmailAddress>
{
    public EmailAddressList()
    { }
}
```

Although at this stage I am not really adding anything to `List<T>` in my custom list, this additional layer serves two purposes. First of all, the API has now become a lot simpler for users unfamiliar with generics as the generics have been taken out of the API completely. So, instead of declaring a list that can hold e-mail address records with the following generics syntax:

```
private List<EmailAddress> emailAddresses = new List<EmailAddress>();
```

you can now declare an address list with this simplified code:

```
private EmailAddressList emailAddresses = new EmailAddressList();
```

This code is a lot easier to read and understand by other developers.

Another benefit of the separate class is that it is easier to add additional methods or even change the underlying type of `EmailAddress` completely, to something like `CollectionBase` for example without breaking existing code.

Once you understand a little how `List<T>` and the classes that inherit from it work, the rest of the code in `GetList` is straight forward:

```
public static EmailAddressList GetList(int contactPersonId)
{
    EmailAddressList tempList = null;
    using (SqlConnection myConnection =
        new SqlConnection(AppConfiguration.ConnectionString))
    {
        // Open connection here. Not shown.
    }
}
```

```

        using (SqlDataReader myReader = myCommand.ExecuteReader())
        {
            if (myReader.HasRows)
            {
                tempList = new EmailAddressList();
                while (myReader.Read())
                {
                    tempList.Add(FillDataRecord(myReader));
                }
            }
            myReader.Close();
        }
    }
    return tempList;
}

```

Once the `ExecuteReader` method returns an open `SqlDataReader`, the code starts looping, creating a new instance of `EmailAddress` with `FillDataRecord` and adding it to the list on each iteration. Creating the `EmailAddress` object is identical to the code in `GetItem`. The initial time it took to create the private helper method has now been compensated for as we can easily reuse it in different situations where we need to create an `EmailAddress` from any object that implements the `IDataRecord` interface.

Inserting, Updating and Deleting Data

While it's nice that you can read a single item or a list of items from the database, the `EmailAddressManager` class really needs some methods to send data to the database. In particular: it needs an `Insert`, an `Update` and a `Delete` method to complete the CRUD acronym I talked about earlier.

In the case of the Contact Person Manager application, both `Insert` and `Update` are handled by the same *upsert* method: `Save()`. This means that the code in the business logic and in the data access layer makes no distinction between these two operations.

The `Save` method in the business layer is straight forward; just as you saw with `GetItem` and `GetList`, it simply forwards the call to the DAL. It stores the return value of the `Save` method (which is the ID of the `EmailAddress` in the database) in the `Id` property:

```

public static int Save(EmailAddress myEmailAddress)
{
    myEmailAddress.Id = EmailAddressDB.Save(myEmailAddress);
    return myEmailAddress.Id;
}

```

Remember, this would be the place to do security checks. For example, you could check if the current user was a `ContentManager` before you'd allow the call to `Save()`. After I show you how `Save()` is implemented in the DAL, I'll show you another variation of `Save` in the business layer that validates the `EmailAddress` before it sends it to the DAL.

The current version in the demo application calls the `Save` method on the associated database class. Let's take a look at the code to see how this all works. This is the `Save()` method in the `EmailAddressDB` class in the data access layer:

```

public static int Save(EmailAddress myEmailAddress)
{
    int result = 0;
    using (SqlConnection myConnection =
        new SqlConnection(AppConfiguration.ConnectionString))
    {
        SqlCommand myCommand = new SqlCommand(
            "spocEmailAddressInsertUpdateSingleItem", myConnection);
        myCommand.CommandType = CommandType.StoredProcedure;

        if (myEmailAddress.Id == -1)

```

```

{
    myCommand.Parameters.AddWithValue("@id", DBNull.Value);
}
else
{
    myCommand.Parameters.AddWithValue("@id", myEmailAddress.Id);
}

myCommand.Parameters.AddWithValue("@email", myEmailAddress.Email);
myCommand.Parameters.AddWithValue("@emailType", myEmailAddress.Type);
myCommand.Parameters.AddWithValue(
    "@contactPersonId", myEmailAddress.ContactPersonId);

DbParameter returnValue;
returnValue = myCommand.CreateParameter();
returnValue.Direction = ParameterDirection.ReturnValue;
myCommand.Parameters.Add(returnValue);

myConnection.Open();
myCommand.ExecuteNonQuery();
result = Convert.ToInt32(returnValue.Value);
myConnection.Close();
}
return result;
}

```

Similar to what you saw before, this code sets up a connection and a command to call the stored procedure `sprocEmailAddressInsertUpdateSingleItem`.

Then, based on whether the ID of the `EmailAddress` is `-1` (a new item) or greater than `-1` (an existing item), it creates a parameter called `@id` and passes it the relevant value: `DBNull.Value` (which translates to `null` in database land) for the new item, or the actual ID for an existing item. This is the "magic number" I talked about earlier in the article. I'll show you in a bit how the stored procedure deals with these values.

The code then continues to setup a few other parameters for the other fields of the `EmailAddress` class: `email`, `emailAddressType` and `contactPersonId`.

It also sets up a `ReturnValue` parameter to catch the return value from the stored procedure. This procedure returns the new ID of an item when it's inserted, or the existing ID in case of an update.

At the end, `ExecuteNonQuery` is called to send the data to the database, the return value is retrieved from the parameter which is then returned to the calling code. It's useful to have the `Save` method return the ID of the item so you can use it in your presentation layer somehow, for example, to redirect the user to a details page showing the new e-mail address.

The final thing you need to look at is the stored procedure that performs either the `INSERT` or the `UPDATE`:

```

CREATE PROCEDURE sprocEmailAddressInsertUpdateSingleItem

    @id int,
    @email nvarchar (100),
    @emailType int,
    @contactPersonId int

AS

    DECLARE @ReturnValue int

```

```

IF (@id IS NULL) -- New Item
BEGIN

    INSERT INTO EmailAddress
    (
        Email,
        EmailType,
        ContactPersonId
    )

    VALUES
    (
        @email,
        @emailType,
        @contactPersonId
    )

    SELECT @ReturnValue = SCOPE_IDENTITY()

END
ELSE
BEGIN

    UPDATE EmailAddress SET
        Email = @email,
        EmailType = @emailType,
        ContactPersonId = @contactPersonId
    WHERE Id = @id

    SELECT @ReturnValue = @id

END
IF (@@ERROR != 0)
BEGIN
    RETURN -1
END
ELSE
BEGIN
    RETURN @ReturnValue
END
GO

```

When the `@id` parameter contains null, the code performs an `INSERT` in the `EmailAddress` table. It then uses `SCOPE_IDENTITY()` to get the new ID of the record that was just inserted.

If `@id` does contain a value, the procedure performs an `UPDATE` for the requested record and then assigns the ID of the record to the `@returnValue` variable which is returned at the end to the calling code.

In both case, the ID of the record is returned to the calling code. There it is caught by the return value parameter, converted to an `int` and returned to the `Save` method in the business logic layer.

Earlier I said that the `Save` method in the business logic layer is also a good place to validate the objects you're saving. To do this, you can implement a method like this:

```

private static void Validate(EmailAddress myEmailAddress)
{
    if (String.IsNullOrEmpty(myEmailAddress.Email))

```

```

    {
        throw new Exception(@"Cannot save an EmailAddress
                               without a valid Email property.");
    }
    if (myEmailAddress.Type == ContactType.NotSet)
    {
        throw new Exception(@"Cannot save an EmailAddress
                               without a valid Type property.");
    }
    if (myEmailAddress.ContactPersonId == -1)
    {
        throw new Exception(@"Cannot save an EmailAddress
                               without a valid ContactPersonId property.");
    }
}

```

This method accepts an instance of `EmailAddress`, validates its properties and then throws an exception when the `EmailAddress` doesn't appear to be valid. In this example, the `EmailAddress` is considered invalid when the `Email` property has not been set, when the `Type` property is still `ContactType.NotSet` or when the `ContactPersonId` contains a value of `-1`.

You call the `Validate` method right before you send the object to the data access layer:

```

public static int Save(EmailAddress myEmailAddress)
{
    Validate(myEmailAddress);
    myEmailAddress.Id = EmailAddressDB.Save(myEmailAddress);
    return myEmailAddress.Id;
}

```

Since the `Save` method can now throw exception when the object is not in a valid state, it's a good idea to implement a public method `Validate` as well that performs the same validation but doesn't throw the exception. You could add an overload for the `Validate` method and have it accept a bool that indicates whether or not you want to throw an exception when the object is in an invalid state. With the public `Validate` method, users of your class can check if the object is valid before they attempt to call `Save`.

The `Delete` mechanism in the `EmailAddressManager` class follows a very similar approach, so I won't show you the code for this. Instead, you're advised to download the entire application at the end of this article and see for yourself.

This concludes the fields, properties, constructors and methods of the three contact record classes. While I only showed you the implementation for the `EmailAddress` class, it should be easy to understand `Address` and `PhoneNumber` as well, as both of these classes follow the exact same pattern.

In the next section, I'll show you the `ContactPerson` class.

The ContactPerson Class - Implementation

The `ContactPerson` class is discussed separately, because it has a few twists that are worth looking at.

In the `Private Variables` and `Public Properties` regions you see code that sets up three lists: one for the `EmailAddresses`, one for the `Addresses` and for the `PhoneNumbers`. Here's the code for the three private backing variables:

```

private AddressList addresses = new AddressList();
private PhoneNumberList phoneNumbers = new PhoneNumberList();
private EmailAddressList emailAddresses = new EmailAddressList();

```

These lists allow you to access the respective contact data collections for a contact person. Earlier you saw how each of these lists inherits from its generics

List<T> counterpart.

There's one more public property that's worth looking at: the `FullName`:

```
public string FullName
{
    get
    {
        string tempValue = firstName;
        if (!String.IsNullOrEmpty(middleName))
        {
            tempValue += " " + middleName;
        }
        tempValue += " " + lastName;

        return tempValue;
    }
}
```

You'll see there's no backing variable for the `FullName` property because the getter makes use of existing variables: `firstName`, `middleName` and `lastName` to build up the complete name of a `ContactPerson`. With this property, showing a contact person's full name is now as easy as accessing this property. Client code no longer needs to bother with the individual fields, and whether the `middleName` actually contains a value or not.

The `GetList` method of `ContactPersonManager` is almost identical to the ones you saw before, so I won't go over it anymore. The `GetItem` method, however, is quite different. In addition to getting its own data, the `GetItem` method is also responsible for getting the associated contact records, like `Addresses` and `Phonenumbers`. However, since there are circumstances where you may not need to associated contact records, the `GetItem` method has a second overload that allows you to control this. Either call `GetItem` with only the ID of the contact person, or call `GetItem` and pass `false` for the second parameter: `getContactRecords`. If you call `GetItem` with the ID and `true` as the value for the `getContactRecord` parameter, you get a fully populated `ContactPerson` object, with all of its contact data collections set up as well.

```
public static ContactPerson GetItem(int id)
{
    return GetItem(id, false);
}

public static ContactPerson GetItem(int id, bool getContactRecords)
{
    ContactPerson myContactPerson = ContactPersonDB.GetItem(id);
    if (myContactPerson != null && getContactRecords)
    {
        myContactPerson.Addresses = AddressDB.GetList(id);
        myContactPerson.EmailAddresses = EmailAddressDB.GetList(id);
        myContactPerson.PhoneNumbers = PhoneNumberDB.GetList(id);
    }
    return myContactPerson;
}
```

The first overloads calls the second, and passes a default value of `false` for `getContactRecords`.

Just as with the `EmailAddressManager` class, the `ContactPersonManager` class calls `ContactPersonDB.GetItem(id)` to get a `ContactPerson` instance from the database. It then checks if the `ContactPerson` returned from the DAL is not null and that `getContactRecords` is `true`. If both are `true`, the code proceeds by assigning values to the three lists for the `Addresses`, `EmailAddresses` and the `PhoneNumbers`. Each of these lists gets a value by calling its associated `GetList` method in the DAL, passing it the ID of the contact person. That way, the `Address` class knows for which contact person it must retrieve address records for example.

At the end of the code, if the `ContactPerson` was found in the database, it is returned to the calling code. Otherwise, `myContactPerson` would contain null indicating to calling code that there is no `ContactPerson` for the requested ID.

The `Save` method of the `ContactPersonManager` class is also capable of saving the associated contact records. It does this by calling `Save` on each contact record in the list when requested:

```
public static int Save(ContactPerson myContactPerson)
{
    using (TransactionScope myTransactionScope = new TransactionScope())
    {
        int contactPersonId = ContactPersonDB.Save(myContactPerson);
        foreach (Address myAddress in myContactPerson.Addresses)
        {
            myAddress.ContactPersonId = contactPersonId;
            AddressDB.Save(myAddress);
        }

        foreach (EmailAddress myEmailAddress in myContactPerson.EmailAddresses)
        {
            myEmailAddress.ContactPersonId = contactPersonId;
            EmailAddressDB.Save(myEmailAddress);
        }

        foreach (PhoneNumber myPhoneNumber in myContactPerson.PhoneNumbers)
        {
            myPhoneNumber.ContactPersonId = contactPersonId;
            PhoneNumberDB.Save(myPhoneNumber);
        }

        // Assign the ContactPerson its new (or existing) ID.
        myContactPerson.Id = contactPersonId;

        myTransactionScope.Complete();

        return contactPersonId;
    }
}
```

First, the `ContactPersonManager` saves the `ContactPerson` by calling a method with the same name in the `ContactPersonDB` class. This is identical to how objects like `EmailAddress` are saved in the database. Then the code continues to save each individual contact record by looping through the respective collections and passing each contact record to the `Save` method in the respective `*DB` class.

```
foreach (EmailAddress myEmailAddress in myContactPerson.EmailAddresses)
{
    myEmailAddress.ContactPersonId = contactPersonId;
    EmailAddressDB.Save(myEmailAddress);
}
```

Before `Save` is called, each contact record gets its `ContactPersonId` property assigned, which is the result of the `Save` method of the `ContactPerson`.

Notice how the entire code block for the method is wrapped inside a `using` block that creates a new `TransactionScope` object. The `TransactionScope` automatically sets up a transaction for all other code that falls within its scope. This is useful to ensure the database remains in a valid state. Whenever an error occurs, for example because the code tries to save an `Address` with an invalid `ContactPersonId`, all previous database actions are rolled back. So, for example, when you save a contact person, then four of its addresses and two email addresses but the code fails on the first `Save` action of a phone number, the transaction for the `ContactPerson`, `Addresses` and `EmailAddresses` is undone.

For this code to work correctly, you need to have the Microsoft Distributed Transaction Coordinator up and running. As of Service Pack 2, it's off by default. Check the following article for more details about configuring MSDTC.

- [Distributed Transaction Coordinator](#)
- [How to Enable MSDTC on the Business Management Server](#)
- [Transactional Remote Receive](#)

If you have the web site and the database on different machines, make sure you configure MSDTC on *both* of them.

The `Delete` method in the `ContactPersonManager` class *always* deletes the associated contact records; there is no point in having an `Address` record in the database without an associated `ContactPerson`. In fact, the database will throw an exception when you try to delete a `ContactPerson` that still has contact records attached to it.

```
public static bool Delete(ContactPerson myContactPerson)
{
    return ContactPersonDB.Delete(myContactPerson.Id);
}
```

The `Delete` method doesn't use a `TransactionScope` object to manage transactions. All contact data records are deleted in the procedure that deletes a contact person:

```
CREATE PROCEDURE sprocContactPersonDeleteSingleItem

    @id int
AS

BEGIN TRAN

DELETE FROM
    Address
WHERE
    ContactPersonId = @id

IF @@ERROR <> 0
BEGIN
    ROLLBACK TRAN
    RETURN -1
END

DELETE FROM
    EmailAddress
WHERE
    ContactPersonId = @id

IF @@ERROR <> 0
BEGIN
    ROLLBACK TRAN
    RETURN -1
END

DELETE FROM
    PhoneNumber
WHERE
    ContactPersonId = @id

IF @@ERROR <> 0
```

```

BEGIN
    ROLLBACK TRAN
    RETURN -1
END

DELETE FROM
    ContactPerson
WHERE
    Id = @id

IF @@ERROR <> 0
BEGIN
    ROLLBACK TRAN
    RETURN -1
END

COMMIT TRAN

```

This code first deletes all contact data based on the incoming ID of the contact person. Once the contact records have been deleted, it also deletes the contact person itself. Notice how after each DELETE statement the code checks for an error, and if necessary rolls the transaction back. This is necessary to avoid records from being deleted when an error occurs later in the stored procedure. Thanks to csharpdev who commented on this at the end of this article, pointing out a big mistake I had in an earlier version of this article.

Wrapping it Up

With the code you have seen so far, we have a nice API to manage `ContactPerson` objects and their associated contact records. With the available classes, you can now create, edit and delete `ContactPersons`, `EmailAddresses`, `Addresses` and `PhoneNumbers` programmatically. The following code snippet shows you how to create a `ContactPerson` and save it in the database:

```

ContactPerson myContactPerson = new ContactPerson();

myContactPerson.FirstName = "Imar";
myContactPerson.LastName = "Spaanjaars";
myContactPerson.DateOfBirth = new DateTime(1971, 8, 9);
myContactPerson.Type = PersonType.Family;

Address myAdress = new Address();
myAdress.Street = "Some Street";
myAdress.HouseNumber = "Some Number";
myAdress.ZipCode = "Some Zip";
myAdress.City = "Some City";
myAdress.Country = "Some Country";
myContactPerson.Addresses.Add(myAdress);

EmailAddress myEmailAddress = new EmailAddress();
myEmailAddress.Email = "Imar@DoNotSpam.com";
myEmailAddress.Type = ContactType.Personal;
myContactPerson.EmailAddresses.Add(myEmailAddress);

PhoneNumber myPhoneNumber = new PhoneNumber();
myPhoneNumber.Number = "555 - 2368";
myPhoneNumber.Type = ContactType.Personal;
myContactPerson.PhoneNumbers.Add(myPhoneNumber);

ContactPersonManager.Save(myContactPerson);

```

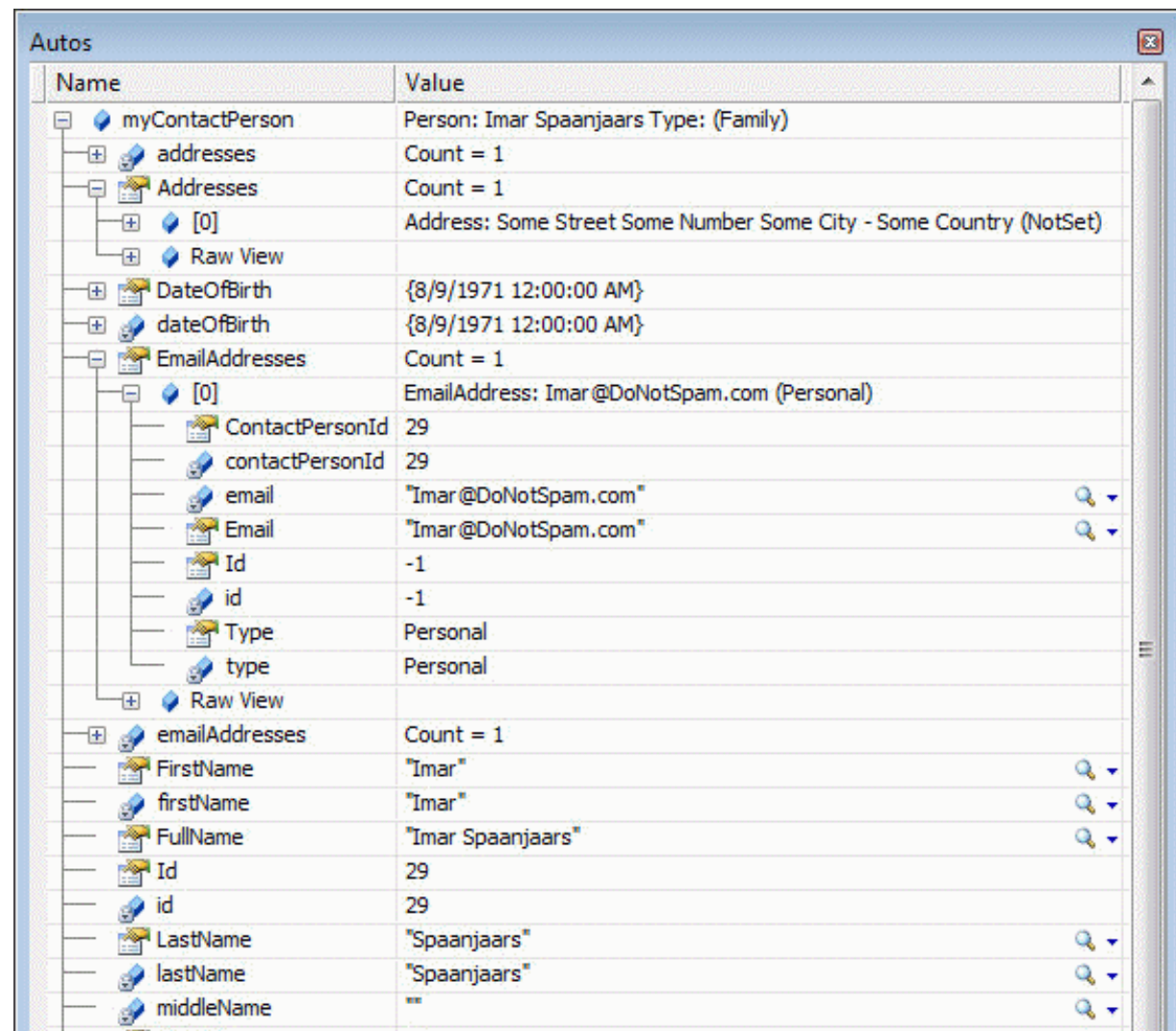
This code creates a new `ContactPerson`, sets a few public properties, and then adds an `Address`, an `EmailAddress` and a `PhoneNumber` instance to the respective collections of the `ContactPerson` object. As soon as `Save` is called, the `ContactPerson` is saved in the database, as well as all of its contact records.

Getting a fully populated `ContactPerson` is now very easy as well. Let's assume the `ContactPerson` in the previous code block got an ID of 26. Getting it from the database, together with all its associated data can be done with a single line of code:

```
myContactPerson = ContactPersonManager.GetItem(26, true);
```

To see what the `ContactPerson` looks like, you can set a breakpoint in your code and then look at the `ContactPerson` object in the watch window. To do this, follow these steps:

1. Clear all existing break points by pressing `Ctrl+Shift+F9`
2. Put your cursor on the line that calls `ContactPersonManager.GetItem(26)` and press `F9`. This sets a breakpoint in the code window. When the code executes, it'll stop here.
3. Hit `F5` to start debugging your application. As soon as the code hits the line above, execution will halt. Notice that the actual line with the breakpoint hasn't executed yet, so you'll need to press `F10` to execute it.
4. Next, right-click the `myContactPerson` variable and choose Add Watch. The variable is added to the Watch window where it looks like this:



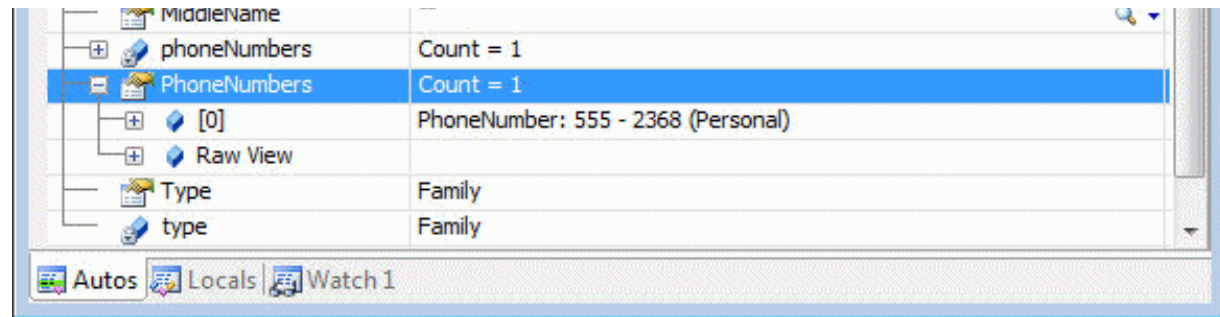


Figure 4 - The Watch Window for the ContactPerson

Notice how I expanded the `Addresses` and `PhoneNumbers` collection one level deep, while I also expanded the first `EmailAddress` in the collection, showing its full details. You may wonder how it's possible that you see these details in the Watch window, instead of the default `Namespace.ClassName` naming scheme that Visual Studio usually uses for custom types. The full details are made available by a `DebuggerDisplay` attribute on the respective classes. For example, the attribute for the `EmailAddress` class looks like this:

```
[
    DebuggerDisplay("Address: {Street, nq} {HouseNumber, nq}
                   {City, nq} - {Country, nq} ({Type})")
]
public class Address
{
```

Within the `DebuggerDisplay`'s constructor you specify a string with placeholders, each one wrapped in a pair of curly brackets. At debug time, these placeholders are replaced with their run-time values. In this example, the `Address` object displays its `Street`, `Housenumber`, `City` and `Country` properties and displays its type in parentheses. The `nq` inside the placeholders stands for no quote and is used to remove the quotes from string variables. Take a look [at this article](#) for a detailed examination of the `DebuggerDisplay` attribute.

While it's certainly useful to manage your contact persons and their contact data programmatically, this isn't always the easiest way to deal with it. It would be much easier if you'd be able to drag and drop a number of controls on a web page, set some properties using wizards, property grids and tasks panes and then be able to manage your contacts in a web browser.

This is exactly what part three of this article series will show you. You'll see how to create pages that display a list of contact persons. This lists is retrieved by calling `GetList()` on the `ContactPersonManager` class through an `ObjectDataSource` control. Each contact person in the list will be updatable and obviously, you'll be able to add new contact persons as well. You'll also be able to add, edit and delete the contact records for each contact person.

Summary

This article showed you how to implement your own business objects in .NET. You saw how to build the classes in the Business Objects and Logic Layers and how to write code for data access in the Data Access Layer. The four classes in the business layer, `ContactPersonManager`, `AddressManager`, `EmailAddressManager` and `PhoneNumberManager` can be used to manage your contact persons and their contact data in any type of application as they are not specific to ASP.NET.

First I showed you how most of the class files in the application are laid out, with a `#region` for each important section of the class. Then you saw how to write the private and public properties of the classes in the business objects layer. Most of the article was spent on discussing the inner workings of the methods in the classes that are responsible for getting data from the database and for inserting, updating and deleting existing records.

At the end of the article you saw a quick example of how you can use these classes programmatically.

Since a programmatic way to manage your contact persons isn't often the easiest solution, [part three](#) of this article series will show you how to use the classes in a web application that mostly uses declarative databinding.

Download Files

- [Source Code of the Demo Application in C#](#). Comes with scripts to create the database
- [Source Code of the Demo Application in Visual Basic .NET](#). Comes with scripts to create the database

Where to Next?

Wonder where to go next? You can read existing comments below or you can post a [comment](#) yourself on this article.

Feedback by Other Visitors of Imar.Spaanjaars.Com

On Saturday 2/17/2007 10:59:49 PM sur said:

I came across part 1 only today and read it from start to finnish. I then noticed Part 2 had just been posted today and read that right through. Im looking forward to part 3. Very well written and i intend to read both articles again. I've read several articles regarding n-tier designs and yours seems to be the best thought out.

There was one line near the end of part 2 which wasnt clear:
myContactPerson = ContactPersonManager.GetItem(26);
Isnt that supposed to read GetItem(26, true)?

Brilliant article again!!!

On Sunday 2/18/2007 12:33:22 AM Imar Spaanjaars said:

Hi sur,

Glad you like the article!! I am working on part three which should be posted on or before Feb 25th.

And yes, good catch. It should indeed be GetItem(26, true). Thanks for that; it's fixed now.

Cheers,

Imar

On Sunday 2/18/2007 3:11:56 PM Tahir said:

Imar,

Good article as expected! One question: ContactPerson's GetItem is overloaded to accept Boolean value and then it retrieves Emails etc. You mentioned that in smart classes one can put lazy loading in property procedures. Is this approach of overloading GetItem similar to that as I guess it achieves the same purpose

Tahir

On Sunday 2/18/2007 3:29:02 PM Imar Spaanjaars said:

Hi Tahir,

Well, not really, actually.

Lazy loading is about postponing loading data, where you postpone loading the values for the properties until you actually need them. As soon as you access a property for the first time, its data is retrieved from the data source. After that, the data remains available.

With the GetItem for the ContactPerson I give a developer the option to choose whether to load all data, or only the contact person's base data. There is no direct way to load the contact records data afterwards.

However, that's the whole idea; you use false as the parameter (or the overload version without the bool option) to get a contact person without any of its related data only when you don't need that data at all.

Hope this clarifies things,

Imar

On Sunday 2/18/2007 3:49:14 PM Tahir said:

Imar,

Yes that was helpful.

As we do not have the smart classes here, in order to mimic the lazy loading will we then write something like:

```
myContactPerson.Addresses = AddressDB.GetList(id);
```

This will load Addresses and then we can use it. Is this correct?

Tahir

On Sunday 2/18/2007 3:54:47 PM Imar Spaanjaars said:

Hi Tahir,

It depends on where you're calling this from.

If your example comes from the BLL, then yes, that is something you could do.

If you're calling this in the presentation layer then you shouldn't use this as it uses the DB classes directly. Instead, you'd use something like this:

```
myContactPerson.Addresses = AddressManager.GetList(myContactPerson.Id);
```

Imar

On Sunday 2/18/2007 4:06:13 PM Tahir said:

Imar,

Yes that is what I meant actually :).

However with this approach there can be one question whether Addresses, Emails etc should be then part of ContactPerson BO because they can are being accessed directly. That however is a matter of choice I guess.

The article is nice as I said before.

Tahir

On Sunday 2/18/2007 6:53:19 PM Imar Spaanjaars said:

Hi Tahir,

Glad you like this article.

Address and ContactPerson are already in the same namespace so they can "legally" use / consume each other.

Making Address "part off" ContactPerson probably isn't a good idea. By making it a nested class, it becomes less clear how to work with instances of Address directly....

Imar

On Sunday 2/18/2007 7:46:21 PM Tahir said:

Hi Imar,

I have discussed about another layer Business Processes previously and in P2P wrox forum as well. I would have created a Process e.g. CreateNewContactProcess that will do the same you did in following code:

```

ContactPerson myContactPerson = new ContactPerson();

myContactPerson.FirstName = "Imar";
myContactPerson.LastName = "Spaanjaars";
myContactPerson.DateOfBirth = new DateTime(1971, 8, 9);
myContactPerson.Type = PersonType.Family;

Address myAdress = new Address();
myAdress.Street = "Some Street";
myAdress.HouseNumber = "Some Number";
myAdress.ZipCode = "Some Zip";
myAdress.City = "Some City";
myAdress.Country = "Some Country";
myContactPerson.Addresses.Add(myAdress);

EmailAddress myEmailAddress = new EmailAddress();
myEmailAddress.Email = "Imar@DoNotSpam.com";
myEmailAddress.Type = ContactType.Personal;
myContactPerson.EmailAddresses.Add(myEmailAddress);

PhoneNumber myPhoneNumber = new PhoneNumber();
myPhoneNumber.Number = "555 - 2368";
myPhoneNumber.Type = ContactType.Personal;
myContactPerson.PhoneNumbers.Add(myPhoneNumber);

ContactPersonManager.Save(myContactPerson);

```

In the CreateNewContactProcess I would do following

```

{
ContactPerson myContactPerson = new ContactPerson();
myContactPerson.FirstName = "Imar";
myContactPerson.LastName = "Spaanjaars";
myContactPerson.DateOfBirth = new DateTime(1971, 8, 9);
myContactPerson.Type = PersonType.Family;
ContactPersonManager.Save(myContactPerson);

Address myAdress = new Address();
myAdress.Street = "Some Street";

```

```

myAddress.HouseNumber = "Some Number";
myAddress.ZipCode = "Some Zip";
myAddress.City = "Some City";
myAddress.Country = "Some Country";
AddressManager.Save(myAddress);

EmailAddress myEmailAddress = new EmailAddress();
myEmailAddress.Email = "Imar@DoNotSpam.com";
myEmailAddress.Type = ContactType.Personal;
EmailAddressManager.Save(myEmailAddress);

PhoneNumber myPhoneNumber = new PhoneNumber();
myPhoneNumber.Number = "555 - 2368";
myPhoneNumber.Type = ContactType.Personal;
PhoneNumbersManager.Save(myPhoneNumber);
}

```

This way each of the business object (ContactPerson, Address, EmailAddress, PhoneNumber) will be cohesive and flexible and UI can call the process which saves us from putting the code behind code-behind. The process can do validations by calling validation funtions of individual business objects (BLL's).

What do you think? I might be generating more work for myself by creating a seperate layer however I feel that it gives more flexibility.

Tahir

On Sunday 2/18/2007 10:41:56 PM Imar Spaanjaars said:

Hi Tahir,

I fail to see how this works or why this would be useful. Is the example correct? What's the point of hardcoding stuff like this:

```

ContactPerson myContactPerson = new ContactPerson();
myContactPerson.FirstName = "Imar";
myContactPerson.LastName = "Spaanjaars";

```

in a separate class?

I am sure you have some way to send information from the UI in this process class, but I can't see how. Either this class has parameters for all the information you want to pass, or it has a strong relation with the presentation layer, in which case I don't really see the difference with, say, the code behind class of a page.

Can you elaborate? It's a bit difficult to post code here, so maybe you can post an example somewhere with more details?

Imar

On Sunday 2/18/2007 11:10:19 PM Tahir said:

Hi Imar,

Well I was not able to explain myself. From the UI the process will get three business objects. The process will then call *ManagerDB.Save for each of them, after doing validations, security checks etc. Note that validations and security etc is also in the BLL's but only called from the process rather than the UI.

I hope I am clearer this time :)

Tahir

On Sunday 2/18/2007 11:14:11 PM Tahir said:

Hi Imar,

The difference between having such calls from UI and a process, IMO, is that one can change the application type between Web, Windows, Web Service etc.

I am converting a Windows based application into a Web based application and such an approach makes life easier. Well don't ask why I am converting as its client's request :).

Tahir

On Monday 2/19/2007 2:37:21 PM Tahir said:

Hi Imar,

I dont see anything wrong with the architecture you presented. Infact this is what many of us use in our real-world project. However, I have one question. Are such architectures Object-Oritned? I am asking this after reading Expert VB.NET Business Objects (CSLA.NET) which you recommended me. Our architecture here seems more procedural than OO.

What are your thoughts on this?

Tahir

On Monday 2/19/2007 2:58:22 PM Imar Spaanjaars said:

Heuh? Why shouldn't this be object oriented? What makes this procedural?

And even of it was, what would it matter?

Imar

On Monday 2/19/2007 3:06:40 PM Tahir said:

Hi Imar,

The reason why I am saying it **may** be procedural is that all we are doing is calling procedures in BLL and DAL. In other words if this **is** OO then what is the difference between this and prcedural architectures?

Why it matters? Well it does'nt at all. Author of even the worse architecture can say 'so what'. I am only discussing a thought!!!

Tahir

On Monday 2/19/2007 3:09:25 PM Imar Spaanjaars said:

Well, it depends on how you look at it.

I'd say that:

```
myContactPerson.Address.Add(new Address());  
myContactPerson.Save();
```

classifies as OO.....

Imar

On Monday 2/19/2007 3:15:48 PM Tahir said:

Yes it will however we are not doing this, we are doing something like:

```
ContactPersonManager.Save(myContactPerson);
```

This is calling a procedure, passing an argument and getting back a result.

What I am discussing with you is 'Architecture'. As I said I have used similar design myself as well but its just a thought.

Tahir

On Monday 2/19/2007 3:18:08 PM Imar Spaanjaars said:

>> just a thought.

Sure....

On Tuesday 2/20/2007 11:40:50 AM Simon said:

Imar, great follow up article.

I was wondering if you could recommend any code tools (like codesmith) that you use to reduce the manual production of Layered Web apps in this style? Or do you tend to do everything manually?

On Tuesday 2/20/2007 1:26:44 PM Imar Spaanjaars said:

Hi Simon,

In-house, in the company I work for, we've created custom code generators that generate much of this code, including all the properties and methods, stored procedures etc. Tools like LLBLGEN and CodeSmith can certainly be used for this as well.

In addition to that, I also use Coderush from DevExpress.com. This tool doesn't generate entire classes for you, but can certainly improve your productivity.

Cheers,

Imar

On Tuesday 2/20/2007 1:58:23 PM Tahir said:

Imar,

The Architecture you presented here is great and works well. Its flexible, team-friendly and simple/logical to understand.

You mentioned on P2P forum in our discussion that the architecture you produce 'depends' on the application, whether its \$2000 or \$20,000. I agree 100%. Can you tell us what scale projects are suitable for this architecture. Also how would you extend/improve this for larger projects. What are the things you will add/change/remove for larger projects.

Tahir

On Tuesday 2/20/2007 6:46:27 PM Imar Spaanjaars said:

Hi Tahir,

The architecture presented in this article scales up to projects of exactly \$17,248.33 and not a penny more..... ;-)

Come on!!!!!! You keep asking questions I cannot answer and worse, you keep asking the same question over and over again. How long is a piece of string?

My last reply in this thread is STILL my take on it: http://p2p.wrox.com/topic.asp?TOPIC_ID=55781&whichpage=4

If you'd agreed 100% you would have understood.....

Imar

On Tuesday 2/20/2007 7:19:08 PM Stan said:

Imar,

Very great article.

I learned very quickly (sorry for the english) and the way you teach the Ntier architecture is very clear I've bought many ASP.net 2.0 books on this topic to understand the concept but yours, once again, is very very great.

However, I have a question about smart class and dumb classes.
Which one is better to use ?

By the way you talked about a Part III, when do think it will be available ????

Stan

On Tuesday 2/20/2007 10:53:23 PM Imar Spaanjaars said:

Hi Stan,

Glad you like the article.

In the intro of this article I said I am aiming to publish part three on or before February 25th. Hopefully, I can make that self-inflicted dead-line.

Cheers,

Imar

On Wednesday 2/21/2007 12:29:39 AM Tahir said:

Hi Imar,

:) Sorry for being a pest. I think what I should do is to write something about my ideas. I will write a short article and if you give me permission I will implement this same application of contacts management using the architecture I am working towards. I dont have a website (yet) so if you dont mind I can send you the article to publish on your website. If not I am planning to build a website for myself and I can publish it there.

Writing about my own ideas is what I should do. This will give me a basis for further discussions with you and other developers.

Tahir

On Wednesday 2/21/2007 4:22:03 AM Glen said:

Firstly, thanks for replying to another question I posted earlier.

Again another great article but would you be able to explain how you would handle a delete of one of the items in a list eg. Address as part of the ContactPersonDB.Save. i.e. Your current code does a foreach save on each address however if an address is deleted no operation would be performed on that address as part of the current save functionality? Is that right?

On Wednesday 2/21/2007 4:37:12 AM Glen said:

Hi,

An amendment to my post earlier, that should read the PersonContactManager.Save not as I have said PersonContactDB.

I have just been thinking about it though, would it be good practice to have a delete flag as part of the business object and setting this when a delete has happend in the UI. As part of the PersonContactManager.Save and iterating thru the Addresses you could then call the AddressManager.Delete when flagged?

On Wednesday 2/21/2007 7:58:18 AM Imar Spaanjaars said:

Hi Glen,

Yeah, there are multiple ways to do this. One would be to override Remove on the Addresses and other lists (you'll need to implement a separate type for that) and then set a Delete flag or immediately delete the item.

Alternatively, you can create a separate list type that holds the addresses. In addition to the standard list of Items, you could keep track of a DeleteItems collection as well. Then saving the Addresses collection would work like this:

```
foreach (Address myAddress in DeleteItems)
{
    myAddress.Delete();
}
```

```
}  
  
foreach (Address myAddress in Items)  
{  
    myAddress.Save();  
}
```

You would still need to override Remove and then move an item from the Items collection into the DeleteItems collection.

Does this help?

Imar

On Wednesday 2/21/2007 8:00:01 AM Imar Spaanjaars said:

Hi Tahir,

Yes, writing about it is a good way to clarify your ideas and thoughts about this.
There are many free Blog sites out there so I am sure you'll find a spot to publish your article on....

Imar

On Wednesday 2/21/2007 10:49:11 AM Tahir said:

Hi Imar,

Hmmm that is a good idea. I will look for a website to blog. Do you recommend any?

Tahir

On Wednesday 2/21/2007 10:49:39 AM Alex said:

Hi Imar,

... if I cannot use "MSDTC", have I to open connection and to call BeginTransaction in BL? After this, have I to pass the connection object to DA classes?

I do not speak english very well, sorry!!!

On Wednesday 2/21/2007 4:00:59 PM antonio said:

How do you avoid circular references if you were to build your application using "smart" objects that manage their own persistence logic?

On Wednesday 2/21/2007 6:51:52 PM Imar Spaanjaars said:

Hi Alex,

Take a look at the SqlTransaction class here:

<http://msdn2.microsoft.com/en-us/library/system.data.sqlclient.sqltransaction.aspx>

Cheers,

Imar

On Wednesday 2/21/2007 6:54:29 PM Imar Spaanjaars said:

Hi antonio,

One way to avoid that is to have the data access code embedded in the object as well. That way, you have an object that holds data, has methods to get and save data, and access the database at the same time.

Alternatively, with a separate data layer, you get methods like this in the DAL:

```
public int Save(int id, string name, string lastName etc etc)
{ }
```

So, for each public property of the business object, you send a .NET simple type in the DAL method.

Does this help?

Imar

On Wednesday 2/21/2007 10:05:51 PM Tahir said:

Hi Imar,

In case of smart classes we will have methods like Load, Save, Remove performing operations on the private properties. However if a client wants to get a list of 'smart' BO then there can be a shared/static method e.g. LoadList that retrieves a list of BOs. The issue that I am thinking is that to load one record it is easy i.e. just filling in the private variables. However how can we return a list of BOs? Secondly I dont want to have a seperate BOList class because that makes data binding with ASP.NET more complex e.g. a datagrid is bound to objectdatasource having TypeName = "BOList" but then the add and delete methods are in BO.

Tahir

On Friday 2/23/2007 4:49:19 AM Glen said:

Hi Imar,

Thanks for the article I am now beginning to put this into practice in my current project. Would you care to comment on the code attributes that are in your code source.

i.e. [DataObjectAttribute()], [DataObjectMethod(DataObjectMethodType.Select, true)] etc

On Saturday 2/24/2007 1:39:10 PM Imar Spaanjaars said:

Hi Glen,

I am glad you like the article. Hopefully, you're making good progress using these principles in your own project?

The next article, part three, talks about DataObjectAttribute and DataObjectMethod, so stay tuned. Hopefully, I can publish the third part this weekend.

Imar

On Monday 2/26/2007 5:49:22 PM sur said:

Just a quick question, in your FillDataRecord methods you use something like:

```
myDataRecord.GetInt32(myDataRecord.GetOrdinal("Id"));
```

Could you also use this instead:
Convert.ToInt32(myDataRecord["Id"]);

or even this:
(int)myDataRecord["Id"];

On Monday 2/26/2007 6:46:12 PM Imar Spaanjaars said:

Hi sur,

Yes, you could do that too, and it would give you the exact same result.

Imar

On Wednesday 3/7/2007 9:44:55 AM Goran said:

Hi Imar!

This is indeed a masterpiece! I love it that you have value objects in one layer (BusinessObject) and the logic in one layer(BusinessLogic). But you should have the layers in different assemblies...

/Goran

On Wednesday 3/7/2007 6:52:14 PM Imar Spaanjaars said:

Hi Goran,

Glad you like the article.

Technically, the different layers *are* already in separate assemblies as each folder in App_Code is compiled to a separate assembly ;-). However, I understand that's not what you meant.

To make it easier for readers of this article to follow along, I decided to use a single web site with an App_Code folder. But I agree with you that for more serious projects it would be necessary to create separate assemblies / projects. That's why I wrote this:

"To continue this separation, I also used different folders inside the App_Code folder of the application. In a larger application you may want to use three separate class libraries instead. The Business Layer class library would then have a reference to the Business Objects and Data Access class libraries, the DAL would also have a reference to the Business objects layer, while the final web site would reference the BO and BLL layers. In the sample application for this article, I decided to keep things simple and store everything in separate folders under App_Code, but the exact same principles would apply if you'd be using class library projects. "

Not everyone has the full version of Visual Studio 2005 which is required to work with separate class libraries in the same solution as the web site. With my current setup, everyone using Visual Web Developer Express Edition and up can follow along with the code and article.

Cheers,

Imar

On Thursday 3/8/2007 8:20:16 AM Saif Khan said:

Great article. From an architect point-of-view this

```
myContactPerson.Addresses = AddressDB.GetList(id);  
myContactPerson.EmailAddresses = EmailAddressDB.GetList(id);  
myContactPerson.PhoneNumbers = PhoneNumberDB.GetList(id);
```

Is a bit chatty. Developers should strive for least connection as possible.

Great articles though Imar.

On Thursday 3/8/2007 9:27:40 AM Goran said:

Hello again Imar!

I've totally missed that section in your article, sorry about that.
And as you said not everyone has full version of Visual Studio 2005.

/Goran

On Thursday 3/8/2007 10:42:59 AM Imar Spaanjaars said:

Hi Saif,

Yeah, that's an interesting point to consider. One way around this is to tie the ContactPerson even further to the contact record classes, and pass an open connection to their respective data access methods. Alternatively, you can have the ContactPerson procedures query the details for the contact person and all of its related data in a single procedure and then use NextResult on a data reader to forward each reader into a constructor of the respective contact data collections.

E.g.

```
SqlDataReader myReader = Get reader with person data  
// fill the person
```

```
// Next fill the Addresses  
myContactPerson.Addresses = AddressDB.GetList(myReader);
```

This way, GetList accepts an open reader and uses it to set up the internal list which is then returned. This is just an example, and there are many other ways to avoid chatty calls in the DAL.

However, since connection pooling is on by default on many Windows systems, this greatly minimizes the impact of opening and closing a connection. Therefore, although the code uses a chatty way to access the data, you probably won't find much performance difference in many situations.

Obviously, your mileage may vary, so it's always good to consider (and test) this. Thanks for bringing it up.

Imar

On Thursday 3/8/2007 3:26:22 PM Rumata said:

Overall, a simple and clean approach. I have a high level question though. I am developing a very large scale web application (millions, hundreds of millions of rows of certain data). Obviously, I am very concerned with performance when picking an architecture. I am actually in the process of rewriting the app and open to various new ideas.

While overall I like your approach and it appeals to me, my concern is performance especially in regards to GetList functionality, where you have to instantiate a class for each record. I think this would impact performance immensely in my case. Do you have any alternative suggestions? What is your take on developing a much larger scale app (where there might not be a lot of classes, but the amounts of data is huge)?

Thanks

On Thursday 3/8/2007 9:34:07 PM Imar Spaanjaars said:

Hi Rumata,

Whatever design you choose, I don't hope you're going to hold all those hundreds of millions of rows in memory at the same time.

Instead, you should look for "paging" or filtering methods in your design. It's a bit hard for me to propose what to do exactly as I don't know how your data looks like and what you want to do with it, but I wouldn't try to have them in memory all the time.

Think of how SQL Server and ADO.NET do it. SQL can contain millions of rows, but when you need them (for example, when exporting data), you don't get all of them in a single object like a DataSet, but instead use a SqlDataReader to get them one by one. That way, you won't bring down your system when you read the data.

Sorry I can't give you a more detailed answer, but it's a bit difficult to suggest something without knowing the full details of your business domain and problem.

Cheers,

Imar

On Friday 3/9/2007 10:54:20 PM Saif Khan said:

Rumata,

As Imar said, it depends on the design. Look at the app in two ways, is this OLTP and reporting.

If reporting, use datasets and paging. There are reappy no objects to reporting so consider webservice so the same reports can then be implemented by several future applications.

If OLTP, No customer will scroll a million records. These days it's always about BI to make quick decisions and most app are trying to incorporate quick summary of data. And above all, like Imar said, why load millions of rows in memory, that's a waste of resources.

Filter the GetList by business parameters like, CustomerID, Date Range, ProductID and so.

Imars work can handle any amount of data, it how use use his design and implementation.

Keep the word "REFACTOR" always at the back of your mind.

Saif

On Friday 3/9/2007 10:56:08 PM Saif Khan said:

Pardon my Grammer.....blame it on Vista. Just kidding.

On Saturday 3/10/2007 8:50:15 PM Jesse Dyson said:

Imar, Thank you for this excellent article! Your article provides the only example I've found so far that is based on Business Objects and leverages the .NET platform. I am using much of your advice in a project I am working on now. I just had one question. In your part 2 article, under "Working with Instances of EmailAddress", you demonstrate a method called GetItem that accepts as a parameter an IPrincipal object. I like this approach since it allows us to delegate the security checks to the Business Logic Layer. How would I go about passing Context. User to this parameter from a DataGrid or DetailsView? Thanks in advance for any assistance.

On Sunday 3/11/2007 11:40:55 AM Imar Spaanjaars said:

Hi Jesse,

Assuming your data bound controls use an ObjectDataSource to get the data, you can use one of the *ing events that fire before the Bll is called (e.g. Selecting, Updating, Inserting and Deleting). Inside these events, you can assign the Context.User to the parameters for the method. Let's take GetList of ContactPerson as an example. First, you need to change the signature:

```
public static ContactPersonList GetList(IPrincipal currentUser)
{ }
```

so it accepts an IPrincipal.

Next, you add an additional parameter to the ODS that selects the data:

```
[SelectParameters]
  [asp:Parameter Name="currentUser" Type="Object" /]
[/SelectParameters]
```

Finally, you handle the Selecting event of the ODS:

```
protected void odsContactPersons_Selecting(
    object sender, ObjectDataSourceSelectingEventArgs e)
{
    e.InputParameters["currentUser"] = Context.User;
}
```

This code assigns the current user to the InputParameters dictionary so eventually it ends up in the currentUser argument of GetList.

Hope this helps,

Imar

On Sunday 3/11/2007 1:21:35 PM Math Random said:

Another great article. It left me with some questions: 1. When you implement security in the BLL using the IPrincipal like you did, can that work with webservices as well? The layered application allows us to create a second or third top layer, but if it NEEDS to pass IPrincipal to each method, isn't that a limitation for webservices? 2. Wouldn't it be better to return an empty list instead of null when there are no records found? I'm not sure about the databinding issues with null values though. 3. How would you program the DAL if you didn't want to use TransactionScope but a 'normal' SqlTransaction? Is that possible when the SQL Commands are run from different methods in different classes? 4. You delete every related record in the Delete method for the ContactPerson, but you also have delete methods on Address and PhoneNumber, so why not loop over the collections and delete them through their own Delete method? Looking forward to your response. I think it's great that you take the time to write this and answer our questions. You're welcome for lunch anytime. Thanks! Oh and about the lazy loading: You could make the property getter check if the addresses are loaded and if not load them. However, that would mean you need a reference from the BO class to a BLL class, and that would mean a circular reference right?

On Sunday 3/11/2007 1:42:12 PM Imar Spaanjaars said:

Hi Math Random,

1. Not necessarily, but it may mean a lot of work. In the book "Expert VB / C# 2005 Business Objects" book by Rockford Lhotka the author presents a way to serialize identities over the network (including web services) allowing you to write security code on both ends of the wire.

2. Yes, I think it would indeed be easier and better to do that. That would be easy to fix. Instead of this:

```
EmailAddressList tempList = null;
```

use this:

```
EmailAddressList tempList = new EmailAddressList();
```

and then don't instantiate a new list inside the HasRows block.

3. One way to do this is to pass your connection object around from method to method.

4. You could do that and it would be a purer way to do things. However, it also means a lot more data access calls. First, you would need to instantiate a ContactPerson, then load its contact data collections and then delete each and every individual object. By letting the spoc handle all of this, you only need a single call.

Regarding the lazy loading: yes, that's the case. However, you might be able to solve this by using an interface although I haven't actually tested this.

Imar

On Tuesday 3/20/2007 10:34:52 AM KUMARAN said:

Hi Imar

Excllent article which help me to create Ntier application.

But I found difficult to create Object datasource and converting into the class file.

Please can you refer me a article to learn creating objectdatasource, converting into class

Hope you could help me.

thanks in advance

regards
kumaran.m

On Tuesday 3/20/2007 12:50:24 PM Imar Spaanjaars said:

Hi KUMARAN,

What do you mean with "create Object datasource and converting into the class file"?

If I understand you correctly, this is not how it works. First, you create the class file, and then connect to that class with an ObjectDataSource. Can you explain your question in more detail?

Imar

On Friday 3/23/2007 6:00:32 AM Saif said:

My advice/opinion: stay away from ODS!

On Friday 3/23/2007 9:11:02 AM Imar Spaanjaars said:

Hi Saif,

In its current form, this is quite a useless statement. You tell people not to use the ODS, but don't offer any alternatives.

Would you care to explain **why** you should stay away from the ODS and what your alternatives are?

Imar

On Sunday 3/25/2007 4:53:07 PM Garry said:

Hey Imar, Although you probably tire of hearing it, I'll repeat "great article!". I need some help with the circular reference issue though. I'm new to OO and have just been exposed to the circular reference concept. From where I sit, I suspect it **could** be a problem at times, but is it always bad? Math Random's question above talked about using the getter of the business object to lazy-load the collection. Even though this is technically a circular reference, is it a bad one? It would work to do it this way, wouldn't it? It seems like the BLL methods would always be static. I could imagine this being a problem if 2 instances referenced each other. You could probably end up in a deadlock, right? But in this case I fail to see the problem apart from a conceptual perspective. Thanks, Garry

On Sunday 3/25/2007 5:54:42 PM Imar Spaanjaars said:

Hi Garry,

Technically, there's often not much stopping you from adding a circular reference. When you try to do it in Visual Studio though, by adding a reference from project A to Project B and then from project B to project A, it won't let you. However, you can bypass this check by browsing to the physical DLL of the other project.

However, this also means that one project is build using an **older version** of the other project. The compiler needs to start with one project. If that project references another, it can't do a full compile of that other project first, so it will use an outdated assembly. This may lead to run-time problems when you later add code to the projects and forget to update all you references and assemblies.

So, it's certainly possible to have circular references in your projects, but it's probably best to always try to avoid them as they can lead to subtle bugs or other issues. In many cases, redesigning your application makes the problem go away.

For some more info on this, take a look here:

<http://www.codeguru.com/forum/archive/index.php/t-332083.html>

At the moment, the site is down, but searching Google for this URL should bring up a cached version.

Also try searching for:

"Adding this project as a reference would cause a circular dependency"

It brings up some interesting results that deal with this topic in more detail.

Cheers,

Imar

On Tuesday 3/27/2007 4:52:16 AM Ashley said:

Hi,

You use the AddressList class to enable the following code simplification:

--note square brackets below are substituting normal generics syntax

```
private List[EmailAddress] emailAddresses = new List[EmailAddress]();
```

you can now declare an address list with this simplified code:

```
private EmailAddressList emailAddresses = new EmailAddressList();
```

I'm struggling to achieve the same code simplification using vb.net. I can create the list class fine, but when I try

to declare the list elsewhere, it still expects the (of T) syntax.

eg: Dim emailAddress as EmailAddressList(Of EmailAddress) = new EmailAddressList(Of EmailAddress)

Am I doing something wrong, or is this a "limitation"/"difference" of vb.net

Thanks
Ashley.

On Tuesday 3/27/2007 7:13:53 AM Imar Spaanjaars said:

Hi there,

How does the declaration for your AddressList in VB look like?

And sorry about the errors; you're not supposed to post anything that looks like HTML, which, unfortunately, includes C# generics syntax... ;-)

Imar

On Tuesday 3/27/2007 7:21:35 AM Ashley said:

My declaration of AddressList in VB looks like..

```
Public Class AddressList(Of Address)
    Public Sub AddressList()
```

```
        End Sub
    End Class
```

On Tuesday 3/27/2007 8:54:28 AM Imar Spaanjaars said:

Hi Ashley

You're missing the inherits part:

```
Public Class AddressList
    Inherits List(Of Address)
```

```
End Class
```

In your code, you just defined a closed generic class, which didn't take out generics from the API.

Cheers,

Imar

On Tuesday 3/27/2007 9:00:24 AM Ashley said:

thanks, that's fixed it.

On Monday 4/2/2007 8:00:46 PM JJ said:

Good article on the designing of the objects and how to go about it. Your business objects are nicely separated; however the catch is that you would be making multiple database connection when you execute the following statement. I am assuming you are making a database connection for each method.

```
private AddressList addresses = new AddressList();  
private PhoneNumberList phoneNumbers = new PhoneNumberList();  
private EmailAddressList emailAddresses = new EmailAddressList();
```

By having each database connection, your objects are nicely encapsulated. I thought about getting all the information with one database connection but that would make the business objects a real pain to design. So in an enterprise setting, what methods are preferred? I know the answers is "it depends" but just want some opinions on the issue.

On Monday 4/2/2007 8:09:57 PM Imar Spaanjaars said:

Hi JJ,

Take a look at my reply to Saif from 3/8/2007 10:42:59 AM.

There are a few ways to do this, including forwarding open connections and forwarding open *DataReaders (where a single sproc retrieves all the data).

However, with connection pooling you can minimize the impact of opening and closing the connection, so having each DAL class perform its own data access will still perform pretty well.

Hope this helps,

Imar

On Tuesday 4/3/2007 2:00:28 PM Milan said:

Hi Imar, great article series. Thanks My q is: If for some reason I want to return ContactPerson list with one Address and one email Address, would you recommend returning them from database immediately with other contact data, or using AddressManager to retrieve one address for each contact?

On Tuesday 4/3/2007 8:11:31 PM Imar Spaanjaars said:

Hi Milan,

Glad you like the articles.

There are a few approaches you can take in your case. One is to create additional methods in the Address class that return a single Address instance based on some criteria and load it in the list.

Another way is to load the entire list into the Addresses collection, and then create a method or property on the Addresses class that is capable of searching the list, and returning the right Address instance to calling code.

Hope this helps,

Imar

On Tuesday 4/10/2007 6:45:38 AM Przemek said:

Hi,

I'm building an app using your tutorial as a template. I tried to compile my code and i get the following error numerous times:

"Error Cannot convert null to 'int' because it is a value type"

This is occuring because I decided to use the Nullable Types solution instead of the -1 solution for the value of NotSet for the Enums. I've read through the link that you gave in your tutorial but I cannot figure out how to code this type of application differently so that I can use Nullable Types.

I know the format is supposed to be 'int ? variable', but I don't know how to code that in the enum.

Please let me know if you know the solution. Thank you.

Przemek

On Wednesday 4/11/2007 9:04:32 AM Imar Spaanjaars said:

Hi Przemek,

You shouldn't use an int?, you should use a nullable version of the enum. E.g.:

ContactType? myContactType = null;

Imar

On Friday 4/13/2007 5:21:30 PM Edmund Ward said:

Hi Imar

Great article (set of articles)! It's such a relief to have a technical article well written. It makes it so much easier to understand.

My question is: if you wanted to store your enums in a table in the database, how would you manage them? Would you have a ContactType business object, ContactTypeManager object and a ContactTypeDB object or is there a simpler way to deal with a "lookup" table?

On Saturday 4/14/2007 6:25:00 AM Przemek Lach said:

I understand your solution; however, how to do implement it in the enum?

Using your example your code looks like this:

```
public enum ContactType
{
    Business = 0,
    Personal = 1,
    NotSet = -1
}
```

How do I make the NotSet = null without getting compile time errors? That is what is happening right now when I run my project.

Thank you.

On Saturday 4/14/2007 9:38:21 AM Imar Spaanjaars said:

Hi Edmund.

Glad you like the series.

I would probably use generics to make a generic ListManager that can retrieve all kinds of lists, like ContactType, Country, Status and so on.

I think I would create a generics class called NameValue (just an example) with a generic K key and a generic V value. You can then instantiate objects of this type using, for example, an int and a string. The int would be the Key in the database (1, 2, 34, whatever) and the string Value would contain the friendly description like Business, Personal and so on.

The ListManager could have methods like GetList which expects an argument that determines the type of list to retrieve. Then GetList would fire the appropriate stored procedure or SELECT statement and return a List of NameValue objects which can be used for databinding.

Finally, the objects itself would store the Id as a property. For example, Address could have a ContactTypeId of type int which maps to the Key (int) field of the NameValue object used to bind, say, a DropDownList.

Hope this helps,

Imar

On Saturday 4/14/2007 9:47:32 AM Imar Spaanjaars said:

Hi Przemek,

The NotSet element in the ContactType enum is used to *emulate* null values. However, since you're now using a true nullable type, you don't need it any longer:

With this code:

```
public enum ContactType
{
    Business = 0,
    Personal = 1
}
```

ContactType? myContactType = null;

you get the same stuff as I had with the NotSet option.

Of course, since it's a nullable type now, you have to treat it a little different and use its properties like Value and HasValue.

Cheers,

Imar

On Sunday 4/15/2007 8:45:37 AM Przemek said:

For some god forsaken reason I cannot get access to the AppConfiguration class. I have a web.config file, I have the proper inclusions in my class but I cannot get it to recognize. When I try to build I get an error at this line:

using (SqlConnection myConnection = new SqlConnection(AppConfiguration.ConnectionString))

Any ideas? Am I possibly missing something painfully obvious?

Thanks

On Sunday 4/15/2007 8:46:45 AM Przemek said:

Sorry about multiple posts. IE glitched out... :-(...???

On Sunday 4/15/2007 9:27:17 AM Imar Spaanjaars said:

Hi Przemek,

Make sure that the casing of the class is the same; as C# is case sensitive.

Also, make sure that if you are using namespaces in your code, both the web page and the AppConfiguration either share the same namespace, or that the page has a using statement for the AppConfiguration's namespace.

Finally, make sure that AppConfiiguration doesn't contain compile errors.

Imar'

On Sunday 4/15/2007 9:57:30 PM Przemek Lach said:

I am casing it correctly: AppConfiguration

I am using namespace in my code. I'm not really sure how to add a namespace in the web.config file. Basically my setup is exactly the same as yours. I can build your application no problem and I do get AppConfiguration no problem. But, not in mine thought.

I'm totally out of ideas. I think I'm screwing something up with the namespace;however, my setup is same as yours so I don't know what else it can be???

Przemek

On Sunday 4/15/2007 10:21:38 PM Imar Spaanjaars said:

Hi Przemek,

You can't add a namespace to the web.config. You should add it to the code.

E.g.

```
namespace SomeName
{
    //Definition of AppConfiguration here
}
```

Then in your classes that use AppConfiguration:

```
using SomeName;
```

But can you expain the problem in more detail? Do you get errors or warnings? What happens when you leave out the implementation from the properties in the AppConfiguration class? Are you then able to see it?

And what if you try:

```
Your.NameSpace.Here.AppConfiguration
```

Can IntellSense see your class?

Finally, do you get IntelliSense on ConfigurationManager in the ConnectionString property? Maybe there's a problem with your code seeing the System.Configuration namespace...

Imar

On Sunday 4/15/2007 10:30:49 PM Przemek said:

That is the problem, the class doesn't come up in IntellSense. So when i try to enter the following line:

```
SqlConnection myConnection = new SqlConnection(AppConfiguration.ConnectionString)
```

I get AppDomain as my first choice, no AppConfiguration.

Now, you talk about creating an AppConfiguration class. Is that something you did in your project? I can't seem to find it.

I thought that in order to access the AppConfiguration class all i would need was a web.config file and a .cs file that has the following at the top:

```
using System;
using System.Data;
using System.Data.Common;
using System.Data.SqlClient;
```

So two files. In my case I have the web.config file an a ThumbnailDB.cs file in a DataAccess folder. The ThumbnailDB.cs file uses the BO namespace and has its own namespace of name.project.Dal.

On Monday 4/16/2007 12:18:48 AM Przemek said:

Imar,

I'm a complete moron. I figured it out.

Thanks anyways.

On Monday 4/16/2007 1:33:15 PM Edmund Ward said:

Hi Imar

Thanks for your earlier reply.

On another issue, I have a product which may qualify for VAT (a tax factor). If I store the factor as a string in the web.config, in which layer should I apply the VAT to the product price? Should it be as part of the presentation, thus simplifying the underlying objects which all deal with prices exclusive of this tax, or should I apply it as part of the business layer:

```
public class MyProduct
{
    private decimal price;
```

```
private bool vatqualifying;

public decimal Price
{
    get
    {
        if (vatqualifying)
        {
            return price * Convert.ToDecimal(WebConfigurationManager.AppSettings["VATFactor"]);
        }
        else
        {
            return price;
        }
    }
}

set { price = value; }
}
```

This means that the price is always stored (in the object and the underlying database) as net of tax, whereas it will be displayed inclusive of tax if appropriate (on a website for example). The trouble with this approach, is that at the product user interface, we have to be careful since we don't know whether the user is entering prices inclusive or exclusive of tax. This makes me wonder whether it might be safer to apply this at the presentation layer. What would you recommend?

Cheers
Edmund

On Monday 4/16/2007 8:28:39 PM Imar Spaanjaars said:

Hi Przemek,

Ha ha, by now you probably figured out that AppConfig is a custom class in the App_Code folder and not built-in in the .NET Framework? ;-)

Glad it's working.

Imar

On Monday 4/16/2007 8:48:29 PM Imar Spaanjaars said:

Hi Edmund,

I would do it as part of the object. That way, the object is in control, so it can determine which amounts are with VAT and which are without. That makes it easier to work with the object.

However, your current design may be a little flawed. Normally, you should be able to set a property more than once, without any side effects. However, in your case, I don't think this is the case. Consider this:

```
myObject.Price = myObject.Price + 0;
```

What do you think happens here when `vatqualifying` is true? Under normal conditions, you'd think this would do nothing to the underlying value. However, in your case, the price is now updated with the VAT included. Do this a couple of times, and your products become too expensive and people will go to your competitors... ;-)

Instead, I'd create a read-only property like `PriceWithVat`, and leave `Price` alone, like this:

```
public decimal Price
{
    get
    {
        return price;
    }
    set
    {
        price = value;
    }
}
```

```
public decimal PriceWithVat
{
    get
    {
        if (vatqualifying)
        {
            return price * Convert.ToDecimal(WebConfigurationManager.AppSettings["VATFactor"]);
        }
        else
        {
            return price;
        }
    }
}
```

Hope this helps,

Imar

On Tuesday 4/17/2007 9:51:55 AM Edmund Ward said:

Hi Imar

Ah, yes, I see what you're saying. The problem with this approach is that I will need to get the users to add/amend prices net of VAT, but possibly view them inclusive of VAT, which might cause confusion. I guess clear labelling is required. Also, this is not really related to the layered model design.

Thanks for your help.

Edmund

On Tuesday 4/17/2007 1:37:42 PM Edmund Ward said:

Hi Imar

As a result of trying to build my app using your model, I keep coming up against questions. So apologies if you're getting tired of these, but here's another one.

I'm building an app for managing used cars. So originally I had a vehicle object and a transaction object, because that is how I expect to store the data in the database. Vehicles might appear in the system as more than one transaction. Vehicle data is stuff like registration plate, VIN, Make, Model, Derivative, etc. Transaction data is stuff like purchase price, value, selling price, mileage, date into stock, date out of stock, delivery date, etc.

When I come to create the presentation layer, however, I'm really mostly interested in Transactions, but will need to have vehicle data to hand too. Would it make sense, therefore, to have a VehicleTransaction object in my BO layer rather than separate Vehicle and Transaction objects? If I have separate objects, how will I bind the vehicle and transaction data to the same control? Or am I barking up the wrong tree here?

Cheers

Edmund

On Tuesday 4/17/2007 8:11:46 PM Imar Spaanjaars said:

Hi Edmund,

Yeah, you could make an intermediate object like VehicleTransaction that acts as the bridge between the two objects. It's important to realize that, unlike relational databases, it's OK to duplicate data in OO objects, as long as the underlying data source stores normalized data. With OO designs like this, it's important to normalize behavior, not necessarily data.

However, you could also create a property of one type on the other, e.g.:

myVehicle.Transaction.Price

for example.

But that all depends on your design and intentions. Hard for me to judge over a few comments on this article.... ;-)

Imar

On Wednesday 4/18/2007 8:59:53 AM Archie said:

Hi Imar

you added collection classes for each business objects using *List T
i'm just wonderin if it would be more effective to use *BindingList T for databinding purposes.Do you think this matters?

I'm actually new here, read that somewhere, just trying to get some insights.

*apparently angel brackets are not allowed

thanks,

Archie

On Wednesday 4/18/2007 10:09:01 AM Edmund Ward said:

Hi Imar

Thanks for your reply. With an intermediate class, would it be independant of the vehicle and transaction classes (i.e. would I need to create a completely new set of private variables and public properties for the intermediate class that encompasses a combination of those of the other two classes, and also have its own BLL and DAL classes?) or would it have properties that somehow use the other classes?

If I went down the other route, having one object as the property of the other (myVehicle.Transaction.Price), would I have to use templates in my presentation layer to specify the properties of the transaction object? I assume that when the ODS hands a list to the GridView, the GridView will only display the vehicle properties. Perhaps I'm wrong, although it doesn't matter too much because I'm intending to use explicit columns anyway to gain control of the column headings.

Your comments are much appreciated.

Cheers

Edmund

On Wednesday 4/18/2007 11:18:58 AM Imar Spaanjaars said:

Hi Edmund,

Yeah, you would probably need to write separate DAL classes.

And indeed: controls like the GridView can't handle nested objects really well.

Imar

On Wednesday 4/18/2007 8:34:17 PM Imar Spaanjaars said:

Hi Archie,

Yes, you could do that too. However, BindingList is more useful in a Windows Forms scenario, while the main topic of this series was Web forms. If your BLL needs to support both environments, BindingList is probably a better choice.

Imar

On Thursday 4/19/2007 11:43:24 AM archie said:

Hi Imar

Thanks for your reply, this series has really moved me a great deal to develop a layered application with your approach/idea (i. e. the use of custom business objects and generics). am currently working on a project now and i've decided to separate the layers in class libraries for scalability. I'm kinda confused on how i'd put up the DAL specifically the way it retrieves the connection string. Unlike the project in this example where there's a web. config to store such info and can be retrieve using "AppConfiguration.ConnectionString", though i could add a connection parameter (i.e GetItem(int ID, string sConn) is there other implementation for this, so i could maintain the signature of the GetItem(int ID) method and makes it look "cleaner"?

On Thursday 4/19/2007 10:08:02 PM Imar Spaanjaars said:

Hi archie,

You could create a static property in the DAL layer that looks in the configuration file just as in my examples.

The location where the DAL is used determines where the connection string is searched for. So, if you use Dal.dll in a web application, it looks in the current web.config by default. For an application, it looks in YourExe.exe. config.

HtH,

Imar

On Tuesday 4/24/2007 6:14:27 AM csharpdev said:

Regarding the contactPerson delete functionality you stated that you did not need to use an ADO transaction, and that: "Since all the code in a stored procedure already automatically runs inside a transaction, there's no need to wrap this code in a TransactionScope object. Either all records are deleted, or none of them are." Thats not right is it? Unless you use ADO's transactions, you would have to explicitly use T-SQL transactions (BEGIN TRANSACTION ROLLBACK, COMMIT, etc.) in order to implement transactions in a stored procedures right? There is certainly no automatic transactions in SP's in my SQL Servers that I am aware of...is this a setting that I can toggle on?

On Tuesday 4/24/2007 7:09:19 PM Adnan said:

hi Imar,
Its a really good & clean article.

I have question about business objects, how we put validation rules for business objects, like if FirstName can have maximum 50 characters, and so on for other properties, i know we have validation controls on GUI for better interaction with user, but will it be better idea that we check bussiness object valid data via some rules, and where to put other business logics in BLL like i have scenrio, a user send me a document with a barcode, if the barcode is valid i want to route this mail a success message using success template and same in case of failure with different kind of failure.

In short my questions are:

Where we should put business objects validation in the architecture?

And how we can display error messages to the user based on if validation fails (Some sort of generic way to handle this)?

Where we should put Business Logic in the architecture ?

Many Thanks & Best Regards,
adnan

On Tuesday 4/24/2007 7:41:21 PM Imar Spaanjaars said:

Hi Adnan,

I mentioned a Validate method in this article, and gave an example on how you can implement and call it.

For a much more detailed implementation of validation in your business objects, check out the CSLA framework by Rockford Lhotka: <http://www.lhotka.net/>

Cheers,

Imar

On Tuesday 4/24/2007 8:19:24 PM Imar Spaanjaars said:

Hi csharpdev,

You are absolutely right. When I initially wrote this, the stored procedure contained only a single DELETE statement. A single statement like that runs in a transaction automatically, so if you can delete record 1 and 2, but not 3, everything is rolled back. I then later updated the procedure, forgot to add transaction code to the stored procedure and update the article :-(Big mistake.

I have updated the database that comes with this article as a download. The procedure now looks like this:

```
BEGIN TRAN
```

```
DELETE FROM
```

```
Address
```

```
WHERE
```

```
ContactPersonId = @id
```

```
IF @@ERROR <> 0
```

```
BEGIN
```

```
ROLLBACK TRAN
```

```
RETURN -1
```

```
END
```

```
DELETE FROM
```

```
EmailAddress
```

```
WHERE
```

```
ContactPersonId = @id
```

```
IF @@ERROR <> 0
```

```
BEGIN
```

```
ROLLBACK TRAN
```

```
RETURN -1
```

```
END
```

```
DELETE FROM
```

```
PhoneNumber
```

```
WHERE
```

```
ContactPersonId = @id
```

```
IF @@ERROR <> 0
```

```
BEGIN
```

```
ROLLBACK TRAN
```

```
RETURN -1
```

```
END
```

```
DELETE FROM  
ContactPerson  
WHERE  
Id = @id
```

```
IF @@ERROR <> 0  
BEGIN  
ROLLBACK TRAN  
RETURN -1  
END
```

```
COMMIT TRAN
```

After each DELETE statement @@ERROR is checked, and if there is an error, the transaction will be rolled back immediately.

I haven't changed the calling code that now has to understand what -1 means which is returned from the sproc in case of an error, but I'll leave that to implementors of this code.

BTW: In SQL Server 2005 you could also use Try/Catch blocks to do pretty much the same, but with a bit cleaner code.

Thanks for catching this; much appreciated. I'll try to update the article ASAP.

Cheers,

Imar

On Tuesday 4/24/2007 8:34:09 PM csharpdev said:

Ok, I thought so, but the rest of your articles are often so good I read that and thought maybe there was some setting I wasn't aware of in SQL Server that would automatically run all SP's as transactions, which might be great for some databases. TRY CATCH looks interesting but it is only in SQL Server 2005. I might use it in the future. Sometimes I resist things that abstract us from whats going on under the hood but offers very little in return, besides maybe slightly cleaner code, though I might use it in the future. Thanks for the articles, I think they are a great contribution to the .NET community. I already have Lhotkas book and I've read parts of it. It's really for projects that are of much larger scale than I work with, but interesting nonetheless. The patters you present here is really more in line with the kind of work I do on a daily basis.

On Thursday 4/26/2007 4:24:22 PM Edmund Ward said:

Hi Imar

Thanks for your replies so far. Next question!! I want to add some AJAX CascadingDropDown control extensions to my application. Whereabouts in your model, would you put the web service? Also the WalkThrough on the ajax.asp.net website suggests using TableAdapters to get the data for the drop downs. These TableAdapters are set up in the App_Code directory. Is this the way to do it for an n-tier model? Is there another way using the business and data layers you have described here? Hope this isn't too far off topic, but I understand your answers better than anyone else's :-o

On Thursday 4/26/2007 6:58:25 PM Imar Spaanjaars said:

Hi Edmund,

Yeah, it's a bit too off-topic for me to answer in detail.

Bottom line is: it's not much different in this case. Put the webservice in the web site and have it call the appropriate methods in the Business Layer.

Cheers,

Imar

On Friday 4/27/2007 11:45:54 PM Suzan said:

Hi Imar,

Thank you very much for your article. Your article is great and it really helps. But I have a problem and maybe you can help me. I have a form with a gridview and it displays all employees until you enter the employee number. Below is EmployeeManager class with 2 public methods. When I ran this code, I got an error message "Cannot implicitly convert type 'EHPersonnel.PeLog.BO.EmployeeList' to 'EHPersonnel.PeLog.BO.Employees' Is there any way to call EmployeeList GetList() method from Employees GetEmployeeByEmpID method? Thanks for your assistance in advance.

Suzan

```
public class EmployeeManager
{

    [DataObjectMethod(DataObjectMethodType.Select, true)]
    public static EmployeeList GetList()
    {
        return EmployeeDB.GetList();
    }

    [DataObjectMethod(DataObjectMethodType.Select, false)]
    public static Employees GetEmployeeByEmpID(string Empl_ID)
```

```
{  
  
    if (string.IsNullOrEmpty(Empl_ID))  
        return GetList();  
  
    else  
        return EmployeeDB.GetEmployeeByEmpID(Empl_ID);  
  
}  
}
```

On Saturday 4/28/2007 10:36:17 AM Imar Spaanjaars said:

Hi Suzan,

You're mixing up the collection and the individual items.

Your GetEmployeeByEmpID is supposed to get a single employee; however when Empl_ID is null or empty, you try to return a list.

Depending on your requirements, there are two ways to fix it:

1. Have GetEmployeeByEmpID return an Employee only; when Empl_ID is null or empty, simply return null.
2. Have GetEmployeeByEmpID return a collection of employees. When Empl_ID is null or empty, return the list. Otherwise, return a list where the first item in the list is the Employee you were looking for.

I think I'd opt for number two; as the method is more consistent; you always return a list, that sometimes only contains a single instance.

However, you could also redesign your app a little and call separate methods: GetEmployeeByEmpID and GetList, depending on the state of the application.

Hope this helps,

Cheers,

Imar

On Monday 4/30/2007 9:23:17 AM Anwer Baig said:

Why u didnt use datasets, to store record set and directly bind those datasets to control like, Gridview, one more thing, is there any harm to use DAAB (Data Access Application Block) to make the DAL.

Regards

Anwer

On Monday 4/30/2007 1:35:56 PM Imar Spaanjaars said:

Hi Anwer,

I am not using DataSets for the same reason I am not using TableAdapters. Check out part one for a discussion on this topic.

IMO, DataSets tend to break the n-tier model and bind the data layer to the presentation layer directly.

And yes, you can use DAAB to build your DAL.

Cheers,

Imar

On Monday 4/30/2007 8:49:25 PM Suzan Cha said:

Hi Imar,

This is in response to your answer. You are right. When I changed my application little bit creating another list that takes a parameter that returns when it's null or empty and it works fine. Thank you very much.

Suzan

On Tuesday 5/1/2007 6:53:09 AM Kurt Gooding said:

Imar.. You genius! Thank you ever so much for providing such an extensible tutorial. I have a question which I have a feeling you might be able to clarify for me. I have created a few validation nmethods with the manager classes, those such as bool CheckDuplicateEmail(EmailAddress myEmail); which simply checks the DB to see if it exists. I am calling this both from the Presentation layer (A button a user can click to check the availability of the email) and within the manager class (when a new contact person is being created). The problem I am facing which I am hoping you can clarify is if I am creating a ContactPerson and the CheckDuplicateEmail returns false (email already in use) how would I display this to the user? The same goes for users without permission when passing the Context.User and they don't have permission, how do I pass an error to the presentation layer? Throwing exceptions seems ugly to me, is returning certain values a clean way? like "-777" and then perform the check in the presentation layer. Or do I simply call the CheckDuplicateEmail in the presentation layer? Once again thank you! Kurt

On Tuesday 5/1/2007 7:04:05 PM Imar Spaanjaars said:

Hi Suzan,

Glad it's all working now.

Cheers,

Imar

On Tuesday 5/1/2007 7:09:20 PM Imar Spaanjaars said:

Hi Kurt,

Glad you like my articles!!

What you could is have Validate return some kind of status, in the form of an enum. E.g.:

```
public enum DataOperationStatus
{
    Success,
    DuplicateRecord,
    ValidationError,
    NotAllowed,
    AndSoOnAndSoOn
}
```

Then have your method return this enum instead of a bool to indicate success. IMO, this is much cleaner than returning magic numbers like -777 which are hard to read and understand.

Your BO could expose a collection of "broken rules". That's how the CSLA framework from Rockford Lhotka does it. When IsValid on the object returns false, you can look at the BrokenRules collection to see what's wrong.

How you implement this, is up to you. BrokenRules could be a simple string collection that you can easily bind to the UI to inform the user what's wrong. Alternatively, it could be a collection of BrokenRule objects that provide more information, like error codes and descriptions.

Hope this helps,

Imar

On Tuesday 5/1/2007 8:18:21 PM Kurt Gooding said:

Music to my ears! I actually started to add a property to the class itself then checking this.. but an enum in this case would definately be a better choice. So thank you yet again this will work perfect. Kurt PS. I noticed you do not have a donations link anywhere.

On Tuesday 5/1/2007 10:38:28 PM Imar Spaanjaars said:

Hi Kurt,

You're right, I don't have a Donation link anywhere. It's a good idea though... I try to make some money from ads on this site, but it's not working very well...

I do have an Amazon Wish Lists for anyone who wants to show his or her appreciation.... ;-)

Cheers,

Imar

On Thursday 5/3/2007 5:52:36 AM Przemek said:

Imar,

I've taken the harder road and I've been trying to learn this N-Layer thing using the nullable type. In one of your answers earlier you mentioned that when using the nullable type, instead of the -1, you have to now use the value and hasValue properties.

Can you show where and how you would modify areas in your code to take this into account.

I've been stuck on this for so long that my next step is going to be to go to doing things the -1 way. I really don't want to do that.

Thank you.

On Thursday 5/3/2007 8:11:19 AM Imar Spaanjaars said:

Hi Przemek,

Take a look here:

[http://msdn2.microsoft.com/en-us/library/2cf62fcy\(vs.80\).aspx](http://msdn2.microsoft.com/en-us/library/2cf62fcy(vs.80).aspx)

It describes how to declare, use and box nullable types. If you have specific questions, I'll be happy to answer them here. Googling for "using nullable types C#" should bring you more useful articles.

However, it's almost impossible for me to post a code solution like the one you're requesting in a comment on this article, so if you need more help, may I suggest you post this in a forum so it's easier to discuss?

Cheers,

Imar

On Thursday 5/3/2007 8:16:42 AM Przemek said:

Imar

Thanks, will do.

PS

Clicked on your ads a couple times to get you some dough ;-)

On Thursday 5/3/2007 8:32:31 PM Suzan Cha said:

Hi Imar, Thanks for your help and I have another problem with DELETE method. I have 4 parameters to pass the delete method. When I click delete button link, it only reads first param corretly and the rest are either 1 or null which is wrong. Here are the methods. Thanks for your help in advance.

Suzan

for DAL

```
public static bool Delete(string Empl_ID, int RatingYear, int RatingMonth, string PType)
{
    int result = 0;
    using (SqlConnection myConnection = new SqlConnection(_ConnectionString))
    {
        SqlCommand myCommand = new SqlCommand("Admin.sp_DeletePESingleItem", myConnection);
        myCommand.CommandType = CommandType.StoredProcedure;
        myCommand.Parameters.AddWithValue("@Empl_ID", Empl_ID);
        myCommand.Parameters.AddWithValue("@RatingYear", RatingYear);
        myCommand.Parameters.AddWithValue("@RatingMonth", RatingMonth);
        myCommand.Parameters.AddWithValue("@PType", PType);

        myConnection.Open();
        result = myCommand.ExecuteNonQuery();
        myConnection.Close();
    }
    return result > 0;
}
```

Here is my BLL

```
[DataObjectMethod(DataObjectMethodType.Delete, true)]
public static bool Delete(PeLogs myPELogs)
{
    return PeLogDB.Delete(myPELogs.Empl_ID, myPELogs.RatingYear, myPELogs.RatingMonth,
myPELogs.PType);
}
```

```
}
```

On Thursday 5/3/2007 10:00:35 PM Imar Spaanjaars said:

Hi Suzan,

Hard for me to tell without seeing the rest of the code, like the controls in the ASPX page.

May I suggest you post this on a forum like <http://p2p.wrox.com>? This is a bit too off-topic for me to investigate and fix for you.

Sorry,

Imar

On Friday 5/4/2007 10:22:35 AM Edmund Ward said:

Hi Imar

I am really pleased with my application now, thanks to your brilliant help. Thank you for your generous time and attention. I have yet another question, however. I want to add an error notification element to my application to let me (or some admin group, say) know when something goes wrong. Say I want to add an email notification element, which layer would actually contain the code to send the email, and which layer would initiate an email send operation? If, for example, there's a database fault in the DAL which causes an exception to bubble up through the layers to the presentation layer, allowing a friendly message to be displayed to the user, the notification mechanism could be started at any layer as part of the exception handling. Is there an accepted wisdom on where this happens? And where would you put the method that actually does the work?

Many thanks for your help.

Cheers
Edmund

On Friday 5/4/2007 7:35:45 PM Imar Spaanjaars said:

Hi there,

There are a number of ways to do this. One common way is to handle all errors in the web application. Simply throw errors when appropriate in the Bll and Dal layers, and catch them globally in the Global.asax file using Application_Error. My book ASP.NET 2.0 Instant Results discusses this in more details.

Alternatively, you may want to Google for Elmah

Cheers,

Imar

On Saturday 5/12/2007 1:13:09 AM Suzan said:

Thanks for your help and I have another question. For Save method, your sample code returns one int value to matching the record. However my database table has 4 fields to make unique ID. Can you help me out? Thanks...

On Saturday 5/12/2007 10:02:11 AM Imar Spaanjaars said:

Hi Suzan,

You can choose whatever you want. You can return a bool to indicate success or failure, you could return the entire object itself, or you could create a Key object that you create yourself. This (composite) Key class simply contains properties for all your unique columns that you assign to in the Save method. Calling code can then access those properties after they called Save.

Imar

On Friday 5/18/2007 8:05:58 PM Rob said:

Does the below statement mean that every BO should have a field/property of type BrokenRulesList? And, after every operation of a BO, you should check to see if BrokenRulesList is DataOperationStatus.Success from the Enum?

On Tuesday 5/1/2007 7:09:20 PM Imar Spaanjaars said:

....

"Your BO could expose a collection of "broken rules". That's how the CSLA framework from Rockford Lhotka does it. When IsValid on the object returns false, you can look at the BrokenRules collection to see what's wrong"

....

Thanks,
Rob

On Saturday 5/19/2007 6:32:56 AM Przemek said:

Imar,

I've did some reading up on nullable types, like you suggested but I still don't understand what is going wrong. Here is a specific example:

I get a runtime error when my app hits this line of code:

```
myThumbnail.Gender = (GenderType)myDataRecord.GetInt32(myDataRecord.GetOrdinal("GenderType"));
```

Gender is an enum that I created and has various values. The runtime error says that the 'specified cast is not valid'. The value that is coming from the database is the number 1.

I do not understand why it is not letting me cast?

Thanks.

On Saturday 5/19/2007 9:49:43 AM Imar Spaanjaars said:

Hi Przemek ,

The problem with this code is not the enum or the nullable type, but the way you get it from the database:

```
myDataRecord.GetInt32(myDataRecord.GetOrdinal("GenderType"))
```

When GenderType is null in the database, you'll get an error when you convert it to an Int32. You'll need to check for DBNull.Value, just as I did with FillDataRecord for the ContactPerson the application.

Cheers,

Imar

On Saturday 5/19/2007 9:55:00 AM Imar Spaanjaars said:

Hi Rob,

Yes, that's how it's done in the CSLA framework. Method like Saves throw an exception when the object is not valid, but you can check the IsValid property before you try to save the object. Then you can use the BrokenRules collection to tell the user what went wrong.

Imar

On Monday 5/21/2007 8:45:50 PM Rob said:

Do any of the 12 examples in you book use the CSLA framework?

Thanks again.

On Monday 5/21/2007 8:47:46 PM Imar Spaanjaars said:

Hi Rob,

Nope, none of them use it. However Rockford Lothka has written a few books about it, as it's his framework.

Imar

On Monday 5/21/2007 8:52:21 PM Imar Spaanjaars said:

BTW: it's important to understand that the CSLA framework uses a lot of inheritance where behavior like this is all implemented in base classes in the framework. You don't have to implement a whole lot in your own BOs to make use of features like BrokenRules.

Imar

On Monday 5/28/2007 1:56:55 AM Przemek said:

Ref: Saturday 5/19/2007 9:49:43 AM Imar Spaanjaars

I do understand what you mean; however, there is a value in the database for this column, it's 1. I was wondering if perhaps the order of the columns in my SELECT has anything to do with it? Or as long as the name that I use, in this case "GenderType" is the same as the name of the column name then I'm ok?

I've tried to use the debugger to see what values I'm actually getting returned here but the debugger has like 50 levels to it and it is difficult for me to figure out exactly what the values are.

Please advise, I'll keep digging.

Thanks.

ShAm

On Monday 5/28/2007 2:46:23 AM Przemek said:

Imar,

Well I figured it out, basically a shot in the dark and it worked. Apparently the datatype in the database has to be 'int', in CANNOT be 'tinyint'. I thought I would save some room in the database by using tinyint but this won't work.

On Monday 5/28/2007 10:25:52 AM Imar Spaanjaars said:

Hi Przemek,

Great; glad it's working.... Thanks for the update.

Imar

Talk Back! Comment on Imar.Spaanjaars.Com

I am interested in what you have to say about this article. Feel free to post any comments, remarks or questions you may have about this article. The Talk Back feature is not meant for technical questions that are not directly related to this article. So, a post like "Hey, can you tell me how I can upload files to a MySQL database in PHP?" is likely to be removed. Also spam and unrealistic job offers will be deleted immediately.

When you post a comment, you have to provide your name and the comment. Your e-mail address is optional and you only need to provide it if you want me to contact you. It will not be displayed along with your comment. I got sick and tired of the comment spam I was receiving, so I have protected this page with a CAPTCHA image now. This means that if you want to leave a comment, you'll need to type in the text you see on the image. If you can't see the text clearly, click the New Image button to get an image with a new text. For more details, see [this news item](#).

If you want to object to a comment made by another visitor, be sure to [contact me](#) and I'll look into it ASAP. Don't forget to mention the page link, or the [QuickDocId](#) of the document.

For more information about the Talk Back feature, check out this [news item](#).

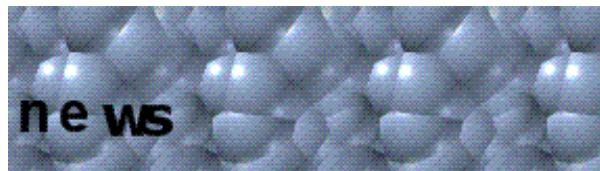
Your Name

Your E-mail Address

Comment

Notify me of replies

Validation Text



Can't see the text on the image? Click the New Image button to display a new image with different text.