Building Collections of Entity Classes using a DataReader

As discussed in the last blog post, it is a best practice to build entity classes. In the last post we filled a DataTable with Category data and then iterated over that DataTable to create a collection of Entity classes. In this blog post we will use a SqlDataReader to fill the Entity classes.

When using a SqlDataReader you must ensure that you close the data reader after you are done with it. You can write a try…catch…finally and close the data reader in the finally block, or you can use the **using** statement. I like the using statement because you do not have to write as much code. In my tests with VS 2010, both ways run just as fast.

I am going to use a new table that I created called Product for this sample. Here is the definition of the Product table. I am switching to another table because I wanted to have a lot of data to run some timing comparisons. I have filled this Product table with over 6200 rows of data. In addition, I wanted some different data types such as DateTime, decimal and boolean to show how to perform conversions and take into account null values in a database.

```
CREATE TABLE Product
(
  ProductId int PRIMARY KEY NONCLUSTERED
                IDENTITY(1,1) NOT NULL,
  ProductName varchar(50) NOT NULL,
  IntroductionDate datetime NULL,
  Cost money NULL,
  Price money NULL,
  IsDiscontinued bit NULL
)
```

You will need to create a class to mimic the columns in the table. Below is a Product class with the corresponding properties to the column in the Product table:

```
C#
public class Product
{
  public int ProductId { get; set; }
  public string ProductName { get; set; }
  public DateTime IntroductionDate { get; set; }
  public decimal Cost { get; set; }
  public decimal Price { get; set; }
  public bool IsDiscontinued { get; set; }
}

Visual Basic
Public Class Product
  Public Property ProductId As Integer
  Public Property ProductName As String
  Public Property IntroductionDate As DateTime
  Public Property Cost As Decimal
  Public Property Price As Decimal
  Public Property IsDiscontinued As Boolean
End Class
```

Below is the code to load a collection of Product data into a collection of Product objects using a data reader.

```
C#
private List<Product> GetProducts()
{
  SqlCommand cmd = null;
  List<Product> ret = new List<Product>();
  Product entity = null;

  cmd = new SqlCommand("SELECT * FROM Product");
  using (cmd.Connection = new SqlConnection(
        "Server=Localhost;Database=Sandbox;Integrated
          Security=Yes"))
  {
    cmd.Connection.Open();
    using (var rdr =
           cmd.ExecuteReader(CommandBehavior.CloseConnection))
    {
      while (rdr.Read())
      {
        entity = new Product();

        // ProductId is a NOT NULL field
        entity.ProductId = Convert.ToInt32(rdr["ProductId"]);
        // Strings automatically convert to "" if null.
        entity.ProductName = rdr["ProductName"].ToString();
        entity.IntroductionDate =
                DataConvert.ConvertTo<DateTime>(
                  rdr["IntroductionDate"],
                   default(DateTime));
        entity.Cost =
                DataConvert.ConvertTo<decimal>(rdr["Cost"],
                  default(decimal));
        entity.Price =
                DataConvert.ConvertTo<decimal>(rdr["Price"],
                  default(decimal));
        entity.IsDiscontinued =
                DataConvert.ConvertTo<bool>(
                  rdr["IsDiscontinued"],
                    default(bool));

        ret.Add(entity);
      }
    }
  }

  return ret;
}

Visual Basic
Private Function GetProducts() As List(Of Product)
  Dim cmd As SqlCommand = Nothing
  Dim ret As New List(Of Product)()
  Dim entity As Product = Nothing

  cmd = New SqlCommand("SELECT * FROM Product")
  Using cnn As SqlConnection = _
  New SqlConnection( _
```

```
      "Server=Localhost;Database=Sandbox;Integrated
         Security=Yes")
    cmd.Connection = cnn
    cmd.Connection.Open()
    Using rdr As SqlDataReader = _
     cmd.ExecuteReader(CommandBehavior.CloseConnection)
      While rdr.Read()
        entity = New Product()

        ' ProductId is a NOT NULL field
        entity.ProductId = Convert.ToInt32(rdr("ProductId"))
        ' Strings automatically convert to "" if null.
        entity.ProductName = rdr("ProductName").ToString()
        entity.IntroductionDate = _
           DataConvert.ConvertTo(Of DateTime) _
             (rdr("IntroductionDate"), DateTime.MinValue)
        entity.Cost = DataConvert.ConvertTo(Of Decimal) _
             (rdr("Cost"), 0D)
        entity.Price = DataConvert.ConvertTo(Of Decimal) _
             (rdr("Price"), 0D)
        entity.IsDiscontinued = _
              DataConvert.ConvertTo(Of Boolean) _
                (rdr("IsDiscontinued"), False)

        ret.Add(entity)
      End While
    End Using
  End Using

  Return ret
End Function
```

The above code is fairly straight forward. Loop through each row and grab each column of data. Convert the data coming from the column into an appropriate value based on the data type. Remember when reading from a DataRow or from a column in the SqlDataReader that the data comes in as an “object” data type. So you must convert it in order to put it into a strongly typed property in your Product object. Of course, you must also handle null values and that is where the DataConvert class comes in.

The DataConvert Class

Whether you use a DataTable/DataSet like in my last blog post or whether you use a DataReader, you will need to check to see if the data read in from the database is a null value. If so, you either need to use Nullable data types in all of your classes, or you need to convert the null to some valid value for the appropriate data type. In the above code I used a class to check for and

convert a null value into a default value for the data. The DataConvert class looks like the following:

```
C#
public class DataConvert
{
  public static T ConvertTo<T>(object value,
    object defaultValue) where T : struct
  {
    if (value.Equals(DBNull.Value))
      return (T)defaultValue;
    else
      return (T)value;
  }
}

Visual Basic
Public Class DataConvert
  Public Shared Function ConvertTo(Of T As Structure) _
    (value As Object, defaultValue As Object) As T
    If value.Equals(DBNull.Value) Then
      Return DirectCast(defaultValue, T)
    Else
      Return DirectCast(value, T)
    End If
  End Function
End Class
```

I used a generic to specify the data type to convert to and then passed in the value from the column and a default value to return if the value is a null.

Summary

In this blog post saw how to create entity classes using a SqlDataReader instead of a Data Table as shown in the previous blog post. In addition you learned how to handle null values by using a DataConvert class.