Creating Collections of Entity Objects using LINQ

As discussed in my last two blog posts you have a variety of ways to create collections of Entity classes. Using a DataSet or DataTable is a little slower than using a DataReader, but in most cases the difference is in milliseconds so in a real world app this difference would not be a killer. For instance, in my sample data I was loading 6,261 records from the Product table discussed in the last blog post and it took 45 milliseconds on average to load those records into an entity collection using a DataTable. It took only 30 milliseconds on average to load the same entity collection using a DataReader. The rendering of that data would probably take longer than that, so you can choose which one you wish to use.

Let's now look at one advantage of using a DataTable. A lot of developers today are used to using LINQ. After loading data into a DataTable you can iterate using a foreach statement, or you can use LINQ to create a collection of entity objects.

Below is a typical entity class that models a Product table in a database:

```
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() 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
```

Reading Data into a Collection using LINQ

Let's now use a LINQ query to iterate over the collection of DataRow objects within a DataTable. In the code below you can see the use of the SqlDataAdapter to fill a DataTable. You now use the AsEnumerable() method on the DataTable to turn the collection of DataRow objects into an enumerable list that can be used in a LINQ statement. In the LINQ statement as you create the new Product object you still use the same DataConvert class to check for valid data and convert into a value that can be stored into each property.

```
C#
public List<Product> GetProducts()
{
  DataTable dt = new DataTable();
  SqlDataAdapter da = null;

  da = new SqlDataAdapter("SELECT * FROM Product",
                          AppSettings.Instance.ConnectString);

  da.Fill(dt);

  var query =
     (from dr in dt.AsEnumerable()
      select new Product
      {
        ProductId = Convert.ToInt32(dr["ProductId"]),
        ProductName = dr["ProductName"].ToString(),
        IntroductionDate =
              DataConvert.ConvertTo<DateTime>(
                dr["IntroductionDate"], default(DateTime)),
        Cost = DataConvert.ConvertTo<decimal>(
                dr["Cost"], default(decimal)),
        Price = DataConvert.ConvertTo<decimal>(
                dr["Price"], default(decimal)),
        IsDiscontinued = DataConvert.ConvertTo<bool>(
                dr["IsDiscontinued"], default(bool))
       });

  return query.ToList();
}


Visual Basic
Public Function GetProducts() As List(Of Product)
  Dim dt As New DataTable()
  Dim da As SqlDataAdapter = Nothing

  da = New SqlDataAdapter("SELECT * FROM Product", _
              AppSettings.Instance.ConnectString)

  da.Fill(dt)

  Dim query = (From dr In dt.AsEnumerable() _
               Select New Product() With { _
  .ProductId = Convert.ToInt32(dr("ProductId")), _
  .ProductName = dr("ProductName").ToString(), _
  .IntroductionDate = DataConvert.ConvertTo(Of _
    DateTime)(dr("IntroductionDate"), DateTime.MinValue), _
  .Cost = DataConvert.ConvertTo(Of Decimal)(dr("Cost"), 0D), _
  .Price = DataConvert.ConvertTo(Of Decimal) _
             (dr("Price"), 0D), _
  .IsDiscontinued = DataConvert.ConvertTo(Of _
        Boolean)(dr("IsDiscontinued"), False) _
  })

  Return query.ToList()
```

```
End Function
```

Summary

In this chapter you learned how to create an entity class and a collection of entity classes using LINQ. When using a DataTable filled with data, LINQ allows you to write less code to create a collection of entities compared to a foreach loop.