Creating Collections of Entity Objects using LINQ and Field Method

Let’s now look at another 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. The DataRow class has an extension method called Field that allows you to check the data and return either a null or the real data value. Of course this means you have to use Nullable types for your properties in your class. Below is the definition of a Product class that uses all Nullable types.

```
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() AsNullable(Of Integer)
  Public Property ProductName() As String
  Public Property IntroductionDate() As Nullable(Of DateTime)
  Public Property Cost() As Nullable(Of Decimal)
  Public Property Price() As Nullable(Of Decimal)
  Public Property IsDiscontinued() As Nullable(Of Boolean)
End Class
```

Reading Data into a Collection using LINQ

Field is a generic method that allows you to pass in the data type you wish to convert the data to if the data within the column is not null. In the code below you can see an example of filling a DataTable with product data, then iterating over the data table using a LINQ query. As you create each new instance of the product class use the Field method to retrieve the data and place it into your nullable property.

Because the Field method is an extension method you will need to add a reference to the System.Data.DataSetExtensions.dll in your project. You could use this method to load properties of a class that does not use Nullable types, but you could not have any null values in your table. This is not a very likely scenario, so you probably want to stick with nullable types.

```
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 = dr.Field<int?>("ProductId"),
        ProductName = dr.Field<string>("ProductName"),
        IntroductionDate =
             dr.Field<DateTime?>("IntroductionDate"),
        Cost = dr.Field<decimal?>("Cost"),
        Price = dr.Field<decimal?>("Price"),
        IsDiscontinued = dr.Field<bool?>("IsDiscontinued")
       });

  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 = dr.Field(Of Nullable(Of _
          Integer))("ProductId"), _
      .ProductName = dr.Field(Of String)("ProductName"), _
      .IntroductionDate = dr.Field(Of Nullable(Of _
          DateTime))("IntroductionDate"), _
      .Cost = dr.Field(Of Nullable(Of Decimal))("Cost"), _
      .Price = dr.Field(Of Nullable(Of Decimal))("Price"), _
      .IsDiscontinued = dr.Field(Of Nullable(Of _
          Boolean))("IsDiscontinued") _
    })

  Return query.ToList()
End Function
```

Summary

In this blog post you learned how to use the Field method on the DataRow class in ADO.NET to help you load up a collection of product objects. When using the Field method you should use .NET nullable data types.

NOTE: You can download the sample code for this article by visiting my website at http://www.pdsa.com/downloads. Select “Tips & Tricks”, then select “Creating Collections of Entities using LINQ and Field Method” from the drop down list.