Binding Source in .Net 2.0

Source	http://www.dotnetspider.com/kb/Article2728.aspx

Introduction

The BindingSource component simplifies the process of binding controls to a data source and provides the following advantages over traditional data binding:

- **Enables design-time binding to business objects.**
- **Encapsulates Currency Manager Functionality and exposes CurrencyManager events at design time.**
- **Simplifies creating a list that supports the IBindingList interface by providing list change notification for data sources that do not natively support list change notification.**
- **Provides an extensibility point for the System.ComponentModel.IBindingList.AddNew method.**
- **Provides a level of indirection between the data source and the control. This indirection is important when the data source may change at run time.**
- **We Can Bind Data Source as Objects, Web Services and Datasets**

The BindingSource component is designed to simplify the process of binding controls to an underlying data source. The BindingSource component acts as both a conduit and a data source for other controls to bind to. It provides an abstraction of your form's data connection while passing through commands to the underlying list of data. Additionally, you can add data directly to it, so that the component itself functions as a data source.

BindingSource Component as an Intermediary

The BindingSource component acts as the data source for some or all of the controls on the form. In Visual Studio, the BindingSource can be bound to a control by means of the DataBindings property, which is accessible from the Properties window.

You can bind the BindingSource component to both simple data sources, like a single property of an object or a basic collection like ArrayList, and complex data sources, like a database table. The BindingSource component acts as an intermediary that provides binding and currency management services. At design time or run time, you can bind a BindingSource component to a complex data source by setting its DataSource and DataMember properties to the database and table, respectively

Sample 1:
The following code example demonstrates how to bind a DataGridView control to a custom type (Object) by using a BindingSource control

```
using System;
```

```
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Windows.Forms;

class Form1 : Form
{
   BindingSource bSource  = new BindingSource();
   DataGridView dgv = new DataGridView();

   public Form1()
   {
       // Bind the BindingSource to the DemoCustomer type.
       bSource.DataSource = typeof(DemoCustomer);

       // Set up the DataGridView control.
       dgv.Dock = DockStyle.Fill;
       this.Controls.Add(dgv);

       // Bind the DataGridView control to the BindingSource.
       dgv.DataSource = bSource  ;
   }
}

// This simple class is used to demonstrate binding to a type.
public class DemoCustomer
{
   // These fields hold the data that backs the public properties.
   DateTime birthDateValue = DateTime.MinValue;
   Guid idValue = Guid.NewGuid();

   // This is a property that represents a birth date.
   public DateTime BirthDate
   {
       get
       {
           return this.birthDateValue;
       }
       set
       {
           if (value != this.birthDateValue)
           {
               this.birthDateValue = value;
           }
       }
   }

   // This is a property that represents a customer ID.
   public Guid ID
   {
       get
       {
           return this.idValue;
       }
   }
}
```

Sample 2:
This form demonstrates using a BindingSource component to bind a list to a DataGridView control. The list does not raise change notifications, so the Reset Item method on the BindingSource is called when an item in the list is changed

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.Common;
using System.Diagnostics;
using System.Drawing;
using System.Data.SqlClient;
using System.Windows.Forms;

// This form demonstrates using a BindingSource to bind
// a list to a DataGridView control. The list does not
// raise change notifications, so the ResetItem method
// on the BindingSource is used.
public class Form1 : System.Windows.Forms.Form
{
   // This button causes the value of a list element to be changed.
   private Button changeItemBtn = new Button();

   // This is the DataGridView control that displays the contents
   // of the list.
   private DataGridView customersDataGridView = new DataGridView();

   // This is the BindingSource used to bind the list to the
   // DataGridView control.
   private BindingSource customersBindingSource = new BindingSource();

   public Form1()
   {
       // Set up the "Change Item" button.
       this.changeItemBtn.Text = "Change Item";
       this.changeItemBtn.Dock = DockStyle.Bottom;
       this.changeItemBtn.Click +=
           new EventHandler(changeItemBtn_Click);
       this.Controls.Add(this.changeItemBtn);

       // Set up the DataGridView.
       customersDataGridView.Dock = DockStyle.Top;
       this.Controls.Add(customersDataGridView);
       this.Size = new Size(800, 200);
       this.Load += new EventHandler(Form1_Load);
   }

   private void Form1_Load(System.Object sender, System.EventArgs e)
   {
       // Create and populate the list of DemoCustomer objects
       // which will supply data to the DataGridView.
       List customerList = new List();
       customerList.Add(DemoCustomer.CreateNewCustomer());
       customerList.Add(DemoCustomer.CreateNewCustomer());
```

```
        customerList.Add(DemoCustomer.CreateNewCustomer());

        // Bind the list to the BindingSource.
        this.customersBindingSource.DataSource = customerList;


        // Attach the BindingSource to the DataGridView.
        this.customersDataGridView.DataSource =
            this.customersBindingSource;
    }

    // This event handler changes the value of the CompanyName
    // property for the first item in the list.
    void changeItemBtn_Click(object sender, EventArgs e)
    {
        // Get a reference to the list from the BindingSource.
        List customerList =
            this.customersBindingSource.DataSource as List;

        // Change the value of the CompanyName property for the
        // first item in the list.
        customerList[0].CompanyName = "Tailspin Toys";

        // Call ResetItem to alert the BindingSource that the
        // list has changed.
        this.customersBindingSource.ResetItem(0);
    }


    [STAThread]
    static void Main()
    {
        Application.EnableVisualStyles();
        Application.Run(new Form1());
    }
}

// This class implements a simple customer type.
public class DemoCustomer
{
    // These fields hold the values for the public properties.
    private Guid idValue = Guid.NewGuid();
    private string customerName = String.Empty;
    private string companyNameValue = String.Empty;
    private string phoneNumberValue = String.Empty;

    // The constructor is private to enforce the factory pattern.
    private DemoCustomer()
    {
        customerName = "no data";
        companyNameValue = "no data";
        phoneNumberValue = "no data";
    }

    // This is the public factory method.
    public static DemoCustomer CreateNewCustomer()
    {
```

```
        return new DemoCustomer();
    }

    // This property represents an ID, suitable
    // for use as a primary key in a database.
    public Guid ID
    {
        get
        {
            return this.idValue;
        }
    }

    public string CompanyName
    {
        get
        {
            return this.companyNameValue;
        }

        set
        {
            this.companyNameValue = value;
        }
    }

    public string PhoneNumber
    {
        get
        {
            return this.phoneNumberValue;
        }

        set
        {
            this.phoneNumberValue = value;
        }
    }
}
```

~~~ End of Article ~~~