INotifyPropertyChanged Interface

Reference

Definition

Namespace: System.ComponentModel

Assembly: System.ObjectModel.dll

Notifies clients that a property value has changed.

C#

```
public interface INotifyPropertyChanged
```

Derived System.Activities.Presentation.Model.ModelItem
System.Activities.Presentation.PropertyEditing.CategoryEntry
System.Activities.Presentation.PropertyEditing.PropertyEntry
System.Activities.Presentation.PropertyEditing.PropertyValue
System.Activities.Presentation.Toolbox.ToolboxCategory
More...

Examples

The following code example demonstrates the how to implement the INotifyPropertyChanged interface. When you run this example, you will notice the bound DataGridView control reflects the change in the data source without requiring the binding to be reset.

If you use the `CallerMemberName` attribute, calls to the `NotifyPropertyChanged` method don't have to specify the property name as a string argument. For more information, see Caller Information.

Replace the code in your Form1 with the following code and then change the namespace to the name of your project. As an alternative, you can name your project with the namespace name below when you create it.

C#

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Runtime.CompilerServices;
using System.Windows.Forms;

// Either change the following namespace to the name of your project,
// or name your project with the following name when you create it.
namespace TestNotifyPropertyChangedCS
{
    // This form demonstrates using a BindingSource to bind
    // a list to a DataGridView control. The list does not
    // raise change notifications. However the DemoCustomer type
    // in the list does.
    public partial class Form1 : Form
    {
        // This button causes the value of a list element to be changed.
        private Button changeItemBtn = new Button();

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

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

        public Form1()
        {
            InitializeComponent();

            // 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(400, 200);
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            // Create and populate the list of DemoCustomer objects
            // which will supply data to the DataGridView.
            BindingList<DemoCustomer> customerList = new
BindingList<DemoCustomer>();
```

```
            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;

        }

        // Change the value of the CompanyName property for the first
        // item in the list when the "Change Item" button is clicked.
        void changeItemBtn_Click(object sender, EventArgs e)
        {
            // Get a reference to the list from the BindingSource.
            BindingList<DemoCustomer> customerList =
                this.customersBindingSource.DataSource as
BindingList<DemoCustomer>;

            // Change the value of the CompanyName property for the
            // first item in the list.
            customerList[0].CustomerName = "Tailspin Toys";
            customerList[0].PhoneNumber = "(708)555-0150";
        }

    }

    // This is a simple customer class that
    // implements the IPropertyChange interface.
    public class DemoCustomer : INotifyPropertyChanged
    {
        // These fields hold the values for the public properties.
        private Guid idValue = Guid.NewGuid();
        private string customerNameValue = String.Empty;
        private string phoneNumberValue = String.Empty;

        public event PropertyChangedEventHandler PropertyChanged;

        // This method is called by the Set accessor of each property.
        // The CallerMemberName attribute that is applied to the optional
propertyName
        // parameter causes the property name of the caller to be substituted
as an argument.
        private void NotifyPropertyChanged([CallerMemberName] String property-
Name = "")
        {
            PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(propertyName));
```

```
        }

        // The constructor is private to enforce the factory pattern.
        private DemoCustomer()
        {
            customerNameValue = "Customer";
            phoneNumberValue = "(312)555-0100";
        }

        // 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 CustomerName
        {
            get
            {
                return this.customerNameValue;
            }

            set
            {
                if (value != this.customerNameValue)
                {
                    this.customerNameValue = value;
                    NotifyPropertyChanged();
                }
            }
        }

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

            set
            {
                if (value != this.phoneNumberValue)
                {
                    this.phoneNumberValue = value;
                    NotifyPropertyChanged();
                }
            }
        }
    }
}
```

Remarks

The INotifyPropertyChanged interface is used to notify clients, typically binding clients, that a property value has changed.

For example, consider a `Person` object with a property called `FirstName`. To provide generic property-change notification, the `Person` type implements the INotifyPropertyChanged interface and raises a PropertyChanged event when `FirstName` is changed.

For change notification to occur in a binding between a bound client and a data source, your bound type should either:

- Implement the INotifyPropertyChanged interface (preferred).
- Provide a change event for each property of the bound type.

Do not do both.

Events

PropertyChanged	Occurs when a property value changes.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8
.NET Framework	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1

Product	**Versions**
UWP	10.0
Xamarin.iOS	10.8
Xamarin.Mac	3.0