LidorSystems Collector

Docking Windows And Tabbed Documents Management

# User Guide

## for LidorSystems.Collector v1.7

# Table of contents

# Introduction

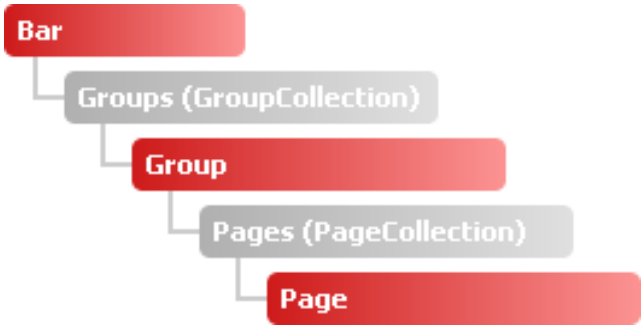
LidorSystems.Collector is .NET class library featuring next generation of windows docking system, bringing a new level of user interaction with your applications. Bundled with many powerful features, it offers you a total control over tool and document window capabilities. With support of WinXP Themes and color schemes and the most precise emulation of Office 2003, VS.NET 2003 and VS.NET 2005 look and feel, it helps you to create attractive and modern user interface in your applications, familiar to the users.

Features include:

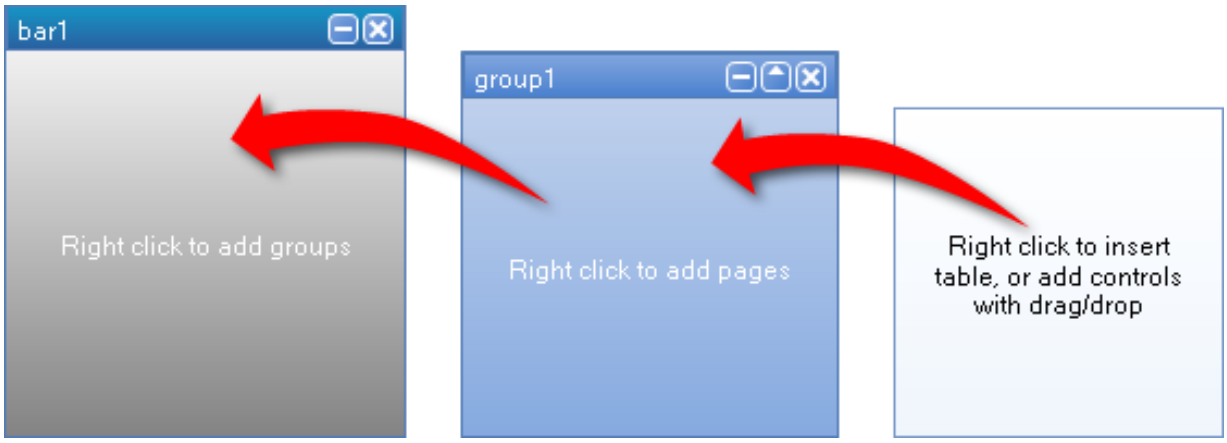
- Design-time support with WYSIWYG functionality
- Rendering interface with five Visual Styles (Default, Classic, Office 2003, VS.NET 2003 and VS.NET 2005)
- Advanced Docking Windows management system
- TabControl and Tabbed MDI solution in one place
- Four style for appearance and functionality of tool and document windows
- Table layout engine for arrangement of controls in tabular format

## Object and event model

**LidorSystems.Collector** class library provides you with extensive object and event model that allows you to configure style and behavior of controls, which enables you to build rich user interfaces in your applications. There are three controls, ordered in hierarchy: **Bar**, **Group** and **Page**.




- **Bar** - The first and highest control in the hierarchy; parent control for Group control; through the Bar you can control the appearance of child controls; can receive only Group controls.
- **Group** - The second control in the hierarchy; parent control for Page control; the look and style for this control can be set independent of Bar control if the **StyleFromParent** property is set to **False** ; can receive only **Page** controls.
- **Page** - The third and basic control in the hierarchy; only Page control can receive all kind of controls. This can be done either by drag&drop controls from Toolbox or using the table layout engine which order controls in tabular format. Table can be managed only in design-time environment.



## Working with GroupCollection

**Bar** control uses **GroupCollection** class to store and work with **Group** controls. With this class you can add, remove and reorder groups programmatically. Any of these operations are followed by events which give you total control in the process. Some of these events are: **GroupAdding**, **GroupAdded**, **GroupRemoving**, **GroupRemoved**, **SelectedGroupChanged** etc.

To add group to existing **Bar** control you need to do the following:

```
[Visual Basic]
' Assuming that Bar control exist
Bar1.Groups.Add(New Group)

[C#]
// Assuming that Bar control exist
this.bar1.Groups.Add(new Group());
```

To remove selected group from **Bar** control you need to do the following:

```
[Visual Basic]
' Assuming that Bar control exist
Bar1.Groups.Remove(Bar1.SelectedGroup)

[C#]
// Assuming that Bar control exist
this.bar1.Groups.Remove(this.bar1.SelectedGroup);
```

By handling the **GroupRemoving** event you can prevent groups to be removed from the **Bar** control:

```
[Visual Basic]
Private Sub Bar1_GroupRemoving(ByVal sender As Object, ByVal ce As _
LidorSystems.Collector.CancelableCollectorEventArgs) Handles Bar1.GroupRemoving
    ce.Cancel = True
End Sub

[C#]
private void bar1_GroupRemoving(object sender,
LidorSystems.Collector.CancelableCollectorEventArgs ce)
{
    ce.Cancel = true;
}
```

**Note** Group control order pages in similar way with use of **PageCollection** class.

## User interface with use of different visual styles

To change the visual style of controls you need to set **VisualStyle** property to one of the following values:

- **Classic** - Used to render controls in Windows Classic control style
- **Office2003** - Used to render controls in Office 2003 control style
- **VS2003** - Used to render controls in Visual Studio .NET 2003 control style
- **VS2005** - Used to render controls in Visual Studio .NET 2005 control style
- **Default** - Used to render controls in Windows XP and Collector control style

Programmatically this can be done with the following code:

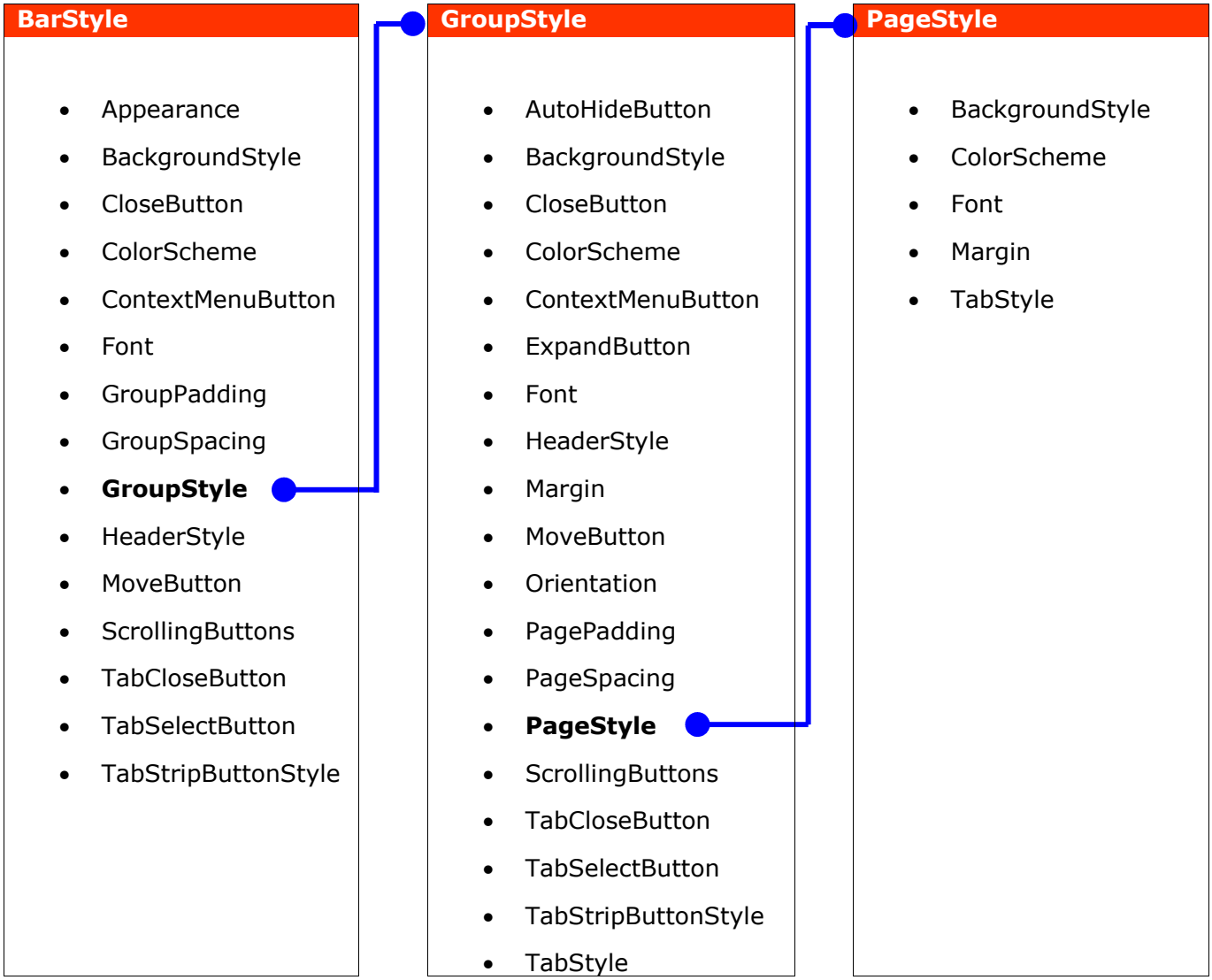
```
[Visual Basic]
' Assuming that Group control exist
Group1.VisualStyle = VisualStyle.Office2003

[C#]
// Assuming that Group control exist
this.group1.VisualStyle = VisualStyle.Office2003;
```

If you want controls to adjust to the current theme and color scheme of windows operating system you need to set **UseTheme** property value to **True**.

With this two properties (**VisualStyle** and **UseTheme**) you can create applications with user interface that is familiar to the users and is well suited in their windows operating systems.

## Custom visual styles

Further more you can create custom user interface by changing the style of controls. Styles, like controls themselves, have hierarchical structure and parent controls can easily change every aspect of appearance and behavior of child controls.



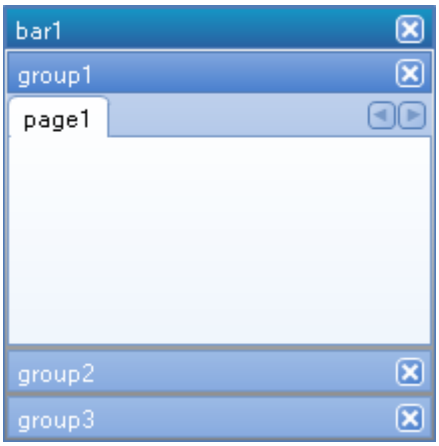

For more freedom in styles, child controls can have their own style. This is set by **StyleFromParent** property value to **False**. The following code fragment show how style can be changed programmatically:

```
[Visual Basic]
' Assuming that Group control exists
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
    ' Change the BackgroundStyle of Group control
    Group1.Style.BackgroundStyle.BackColor = Color.Red
    Group1.Style.BackgroundStyle.BackFadeColor = Color.White
    Group1.Style.BackgroundStyle.FillStyle = FillStyle.ForwardDiagonal

    ' Set orientation of tabs in tabstrip to the right and show only close button
    Group1.Style.Orientation = TabOrientation.Right
    Group1.Style.TabCloseButton = True
End Sub


[C#]
// Assuming that Group control exists
private void Form1_Load(object sender, System.EventArgs e)
{
    // Change the BackgroundStyle of Group control
    this.group1.Style.BackgroundStyle.BackColor = Color.Red;
    this.group1.Style.BackgroundStyle.BackFadeColor = Color.White;
    this.group1.Style.BackgroundStyle.FillStyle = FillStyle.ForwardDiagonal;

    // Set orientation of tabs in tabstrip to the right and show only close button
    this.group1.Style.Orientation = TabOrientation.Right;
    this.group1.Style.TabCloseButton = true;
}
```

To speed up the design process of user interface there are 21 predefined color schemes from which you can start. After you choose color scheme with changes in the style of every control you can make custom user interfaces.

## Appearance

With use of **Appearance** property of **BarStyle** class you can change the overall look and behavior of all child controls:




**Vertical** - Groups are displayed with headers in vertical order within Bar control. All Groups except the selected one is collapsed. This is very useful if you like to achieve the look of Outlook 2003 Navigation Pane.


**Horizontal** - Groups are displayed with tabs in one tabstrip. In this way you will create two level tabs which gives you more power in organization of documents. With this appearance Groups act very similar to Page control, using selection and scrolling buttons, and **TabMode** and **TabStrip** property.


**Selection** - Similar to Horizontal appearance, only one group at the time with the contents is open and editable and the other groups are hidden. The text and image of selected group replaces text and image in the header of Bar control. To select another group use the navigation buttons in the header. This appearance is very similar to the Task Pane in all Office applications.


**ToolWindow** - This appearance emulates the docking functionality of Visual Studio .NET. All Groups are visible and have headers. Windows can be docked anywhere in the container and can be represented by tabs with four types of orientation.

# Docking windows management system

With docking windows users can customize overall look of their application in runtime, getting more usable space. **LidorSystems.Collector** has advanced docking windows management system with which you can:

- Dock controls to any side of the container
- Create floating windows
- Autohide controls to one side of the Form with animation
- Create tabbed docking windows contained in other docked windows
- Add docking permission to controls

## Working with control state

Controls in LidorSystems.Collector library can have one of the following three control states:

- **Static** – Control is contained in the container and cannot be moved or dock.
- **Docked** – Control is docked to one side of container.
- **Floating** – Control represent floating window.

**ControlState** property gives you information about current state of the control. During runtime, control that is docked can become floating window and vice versa. Static controls act like normal controls except that they cannot be moved to become docked or floating during runtime. Normally, this can be overridden programmatically by changing the state of static control.

## Working with docking windows

To dock a window, you simply drag the header of Bar or Group control or drag the tab of Group or Page control, depending of current appearance and hierarchical structure of controls, and move the cursor to the windows edge in the application. When a docking operation can be performed, the drop zone is shown with shape representing the location where window can be dropped.

There are several properties which controls the behavior of controls while performing drag&drop operations, they are:

- **AcceptType** – Determines what type of object can be accepted by the control
- **AllowDocking** – Determines the dock side on which controls can be docked
- **AllowDrag** – Determines whether the control can be dragged
- **AllowFloat** – Determines whether the control can become floating window
- **AllowReorder** – Determines whether the control supports reordering of child controls
- **ControlType** – Determines the type of control
- **DockHintStyle** – Determines the shape of drop zone
- **DropAllowed** - Determines whether the control can accept other controls during drag&drop operations

The following text describes how you can dock windows with use of drag&drop operations. Control which can be dragged will be named as *dragobject*.

To begin a drag&drop operation you need to press and hold left mouse button on *dragobject* and move the cursor in any direction to detach *dragobject* from container. This will raise the

**BarDragging**, **GroupDragging** or **PageDragging** event (depending of the type of drag object). Events is raised in the following manner:

- **BarDragging** event is raised only when you start drag operation from the header of Bar control.
- **GroupDragging** event is raised when you start drag operation from the header of the Group control or from the group tab when the Bar control has horizontal appearance.
- **PageDragging** event is raised when you start drag operation from the page tab.

Drag operation can be halted if you set the **AllowDrag** property value to **False**. Also, if you handle these events you can prevent drag operation to continue. In the following code is presented GroupDragging event; this applies also for other two events:

```
[Visual Basic]
Private Sub Group1_GroupDragging(ByVal sender As Object, _
   ByVal ce As LidorSystems.Collector.CancelableCollectorEventArgs) Handles
Group1.GroupDragging
   ce.Cancel = True
End Sub

[C#]
private void group1_GroupDragging(object sender,
LidorSystems.Collector.CancelableCollectorEventArgs ce)
{
   ce.Cancel = true;
}
```

When you move *dragobject* the mouse cursor will change to represent the object that is dragged (there are three different cursors for each control). Also drop zone is shown with shape set from **DockHintStyle** property. This property can have three different values: **RubberBand**, **TranslucentClone** and **TranslucentFill**.

While moving the *dragobject* you will notice that drop zone will change showing the destination where *dragobject* can be dropped. During this process the **GroupDragOver** event is raised with parameters holding the *dragobject*, mouse position and permission to drop the object to the destination. You can handle this event to check the type of *dragobject* and whether it is allowed to drop it to the destination. The following code check whether the *dragobject* is Group control and if it is, it displays a message in the caption of the Form describing the object, whether can be dropped to the destination, and the location where it will be dropped. You can handle this event to restrict locations where *dragobject* can be dropped.

```
[Visual Basic]
Private Sub Group1_GroupDragOver(ByVal sender As Object, _
    ByVal de As LidorSystems.Collector.CollectorDragEventArgs) Handles
Group1.GroupDragOver

    If (TypeOf de.CollectorObject Is Group) Then
        Me.Text = CType(de.CollectorObject, Group).Text + " - AllowDrop (" +
de.AllowDrop.ToString() _        + "), Location (" + de.X.ToString() + ", " +
de.Y.ToString() + ")"
    End If

End Sub

[C#]
```

```
private void group1_GroupDragOver(object sender,
LidorSystems.Collector.CollectorDragEventArgs de)
{
   if (de.CollectorObject is Group)
      this.Text = ((Group)de.CollectorObject).Text + " - AllowDrop (" +
de.AllowDrop.ToString() + "),      Location (" + de.X.ToString() + ", " +
de.Y.ToString() + ")";

}
```

**Note** The type of **CollectorObject** can be either Bar, Group or Page control.

Depending of the drop zone (whether is CollectorObject or some other control), *dragobject* can be filled or docked to one side of the target control. Because of strong hierarchy among Collector controls *dragobject* can be accepted by control which is one level up in hierarchy, this means that:

- **Bar** control can only be docked to one side of the target control.
- **Group** control can be accepted by Bar control and another Group control. If the drop zone is Bar control then reordering of groups can occur; if the drop zone is Group control then pages from the source Group will be merged with pages in the targeted Group control.
- **Page** control can be accepted only by Group control. By dragging Page control you can reorder tabs.

In the first place to designate your drop zone you must set the **DropAllowed** property of the target control to **True**.

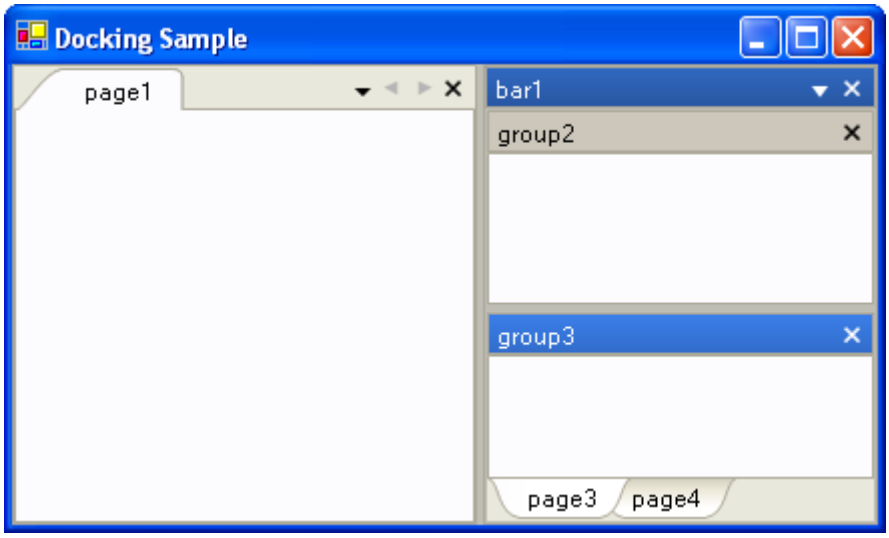
When the *dragobject* is dropped to the drop zone, the following events are raised: **BarDropped**, **GroupDropped** and **PageDropped** (depending of the type of *dragobject*). The following code check whether the *dragobject* is Group control and if it is displays a message in the caption of the Form describing the Group control:

```
[Visual Basic]
Private Sub Group1_GroupDropped(ByVal sender As Object, _
   ByVal de As LidorSystems.Collector.CollectorDropEventArgs) Handles
Group1.GroupDropped
   If (TypeOf de.CollectorObject Is Group) Then
      Me.Text = CType(de.CollectorObject, Group).Text
End Sub

[C#]
private void group1_GroupDropped(object sender,
LidorSystems.Collector.CollectorDropEventArgs de)
{
   if (de.CollectorObject is Group)
      this.Text = ((Group)de.CollectorObject).Text;
}
```

## Working with documents and tool windows

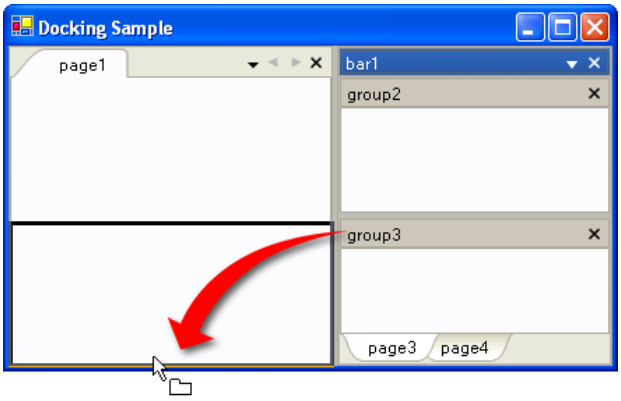
While designing your user interface, you may decide that some windows will act like documents and others like tool windows. Document windows most of the time are represented by tabs and they form tabbed user interface. Tool windows are tab-docked or docked to one side of the parent window. In LidorSystems.Collector with use of **ControlType** property you can decide what type of window you will create.

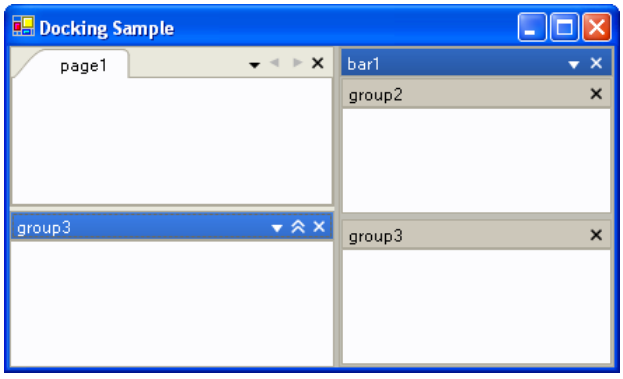
The following example demonstrates the use of drag&drop operations with document and tools windows and different type permissions. This example use **Bar**, **Group** and **Page** controls with **VisualStyle** set to **VS2005** and **UseTheme** set to **True**:




On the left side is Group control without header and with **Dock** property value set to **Fill**. Group can accept only documents because **AcceptType** property value is set to **Documents**.
On the right side is Bar control with **Appearance** property value set to **ToolWindow** and two contained groups with pages. All controls here are tool windows because they have **ControlType** property value set to **ToolWindow**. Also, the style of **Bar** control is controlling the appearance and behavior of groups. The **Style->GroupStyle->Orientation** property is set to **Bottom** which align pages to the bottom side.

While dragging the page3 control we can notice some docking restrictions for this control. Because page3 is tool window it cannot be accepted by the group1 control (which can accept only documents windows). So the page3 control can be docked to any side of the Form or it can create floating window. In this example page3 is docked to the bottom side.



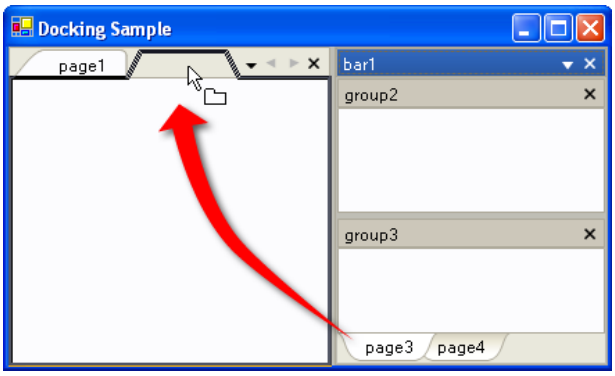



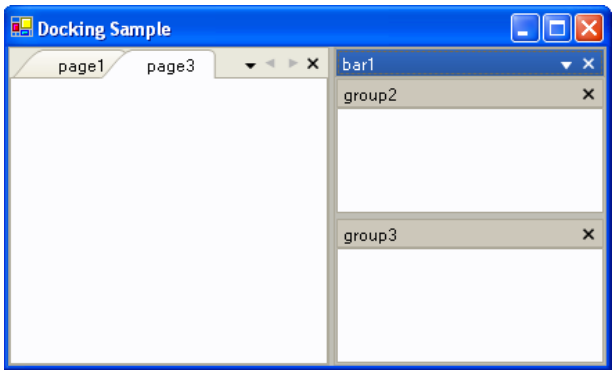

**Note** To create more usable space for documents, you can click the expand button on header of newly created group and the group will collapse.

For group1 (documents) to accept tool windows, we need to change **AcceptType** of this control to **Both**:

```
[Visual Basic]
Group1.AcceptType = AcceptType.Both

[C#]
this.group1.AcceptType = AcceptType.Both;
```

Now, when we try to drag&drop page3 from the group3 to the group1 control, drop zone will change showing area where control can be dropped:

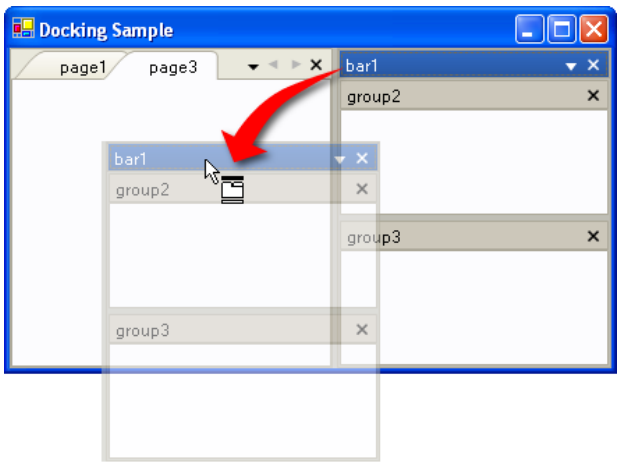


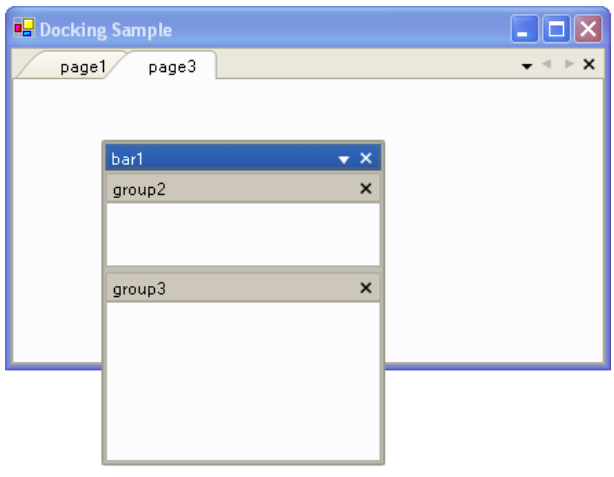




You can change the **DockHintStyle** property in a way that for some controls when they are dragged, drop zone will be represented by reversible frame and for others with translucent clone or translucent fill.

```
[Visual Basic]
Bar1.DockHintStyle = DockHintStyle.TranslucentClone

[C#]
this.bar1.DockHintStyle = DockHintStyle.TranslucentClone;
```

When we try to drag the bar1, drop zone will have translucent image representing the current control that is dragged. If we drop bar1 in free space and if **AllowFloat** property is set to **True**, floating window will be created.



Position and size of floating window can be set with **FloatingLocation** and **FloatingSize** properties. If this properties are not set, then the floating window will receive parameters from the control that becomes floating.

To create floating window programmatically you can use the **MakeControlFloat** function:

```
[Visual Basic]
' Assuming that Group control exists
Group1.MakeControlFloat()

[C#]
// Assuming that Group control exists
this.group1.MakeControlFloat();
```

## Working with AutoHide windows

Another way to create more usable working space is to make dockable windows hidden. To do this, click on the **AutoHideButton** from the header of Group control. This will cause dockable window to slide to the edge of the Form and create AutoHideBar control (if it is not already created). AutoHideBar control can be created only in runtime and it's used for holding Group controls which are in autohide state. When groups are autohidden they can show up if you move mouse cursor over tabs in the AutoHideBar. Groups will slide off, which can be followed by animation. To return the group in normal state click again on the same button.

To make Group control autohide programmatically you can use the **AutoHide** property:

```
[Visual Basic]
' Assuming that Group control exists
Group1.AutoHide = True

[C#]
// Assuming that Group control exists
this.group1.AutoHide = true;
```

After that the **AutoHideBar** control is created and by default it will push other controls (in the Form) beneath, in order to create maximum usable space for auto-hidden controls. In your layout if you have toolbars, status bars and controls that need to stay in their positions, you need to create boundary in which groups can become auto-hidden. This can be done by setting the **AutoHideOuterControl** property of the Group that becomes auto-hidden**,** to the control that will act as boundary or by handling the **AutoHideChanged** event of the Group that becomes auto-hidden and set the **OuterControl** property of the AutoHideBar control to the control that will act as boundary. From the picture bellow you can see how this can be done.

**Fig. A** - Creation of AutoHideBar control by default

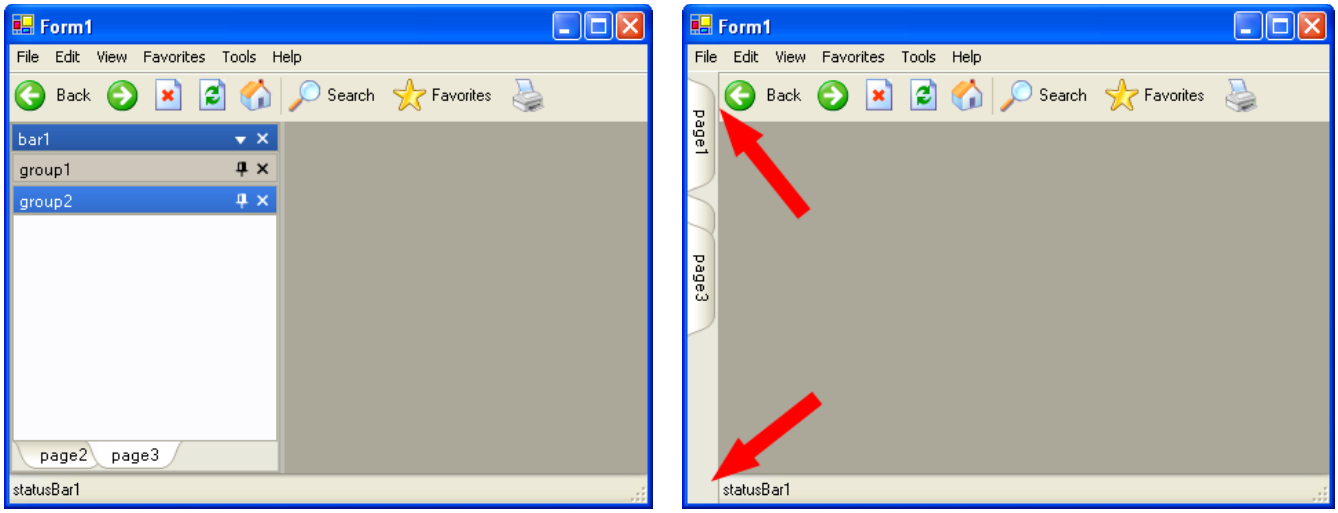
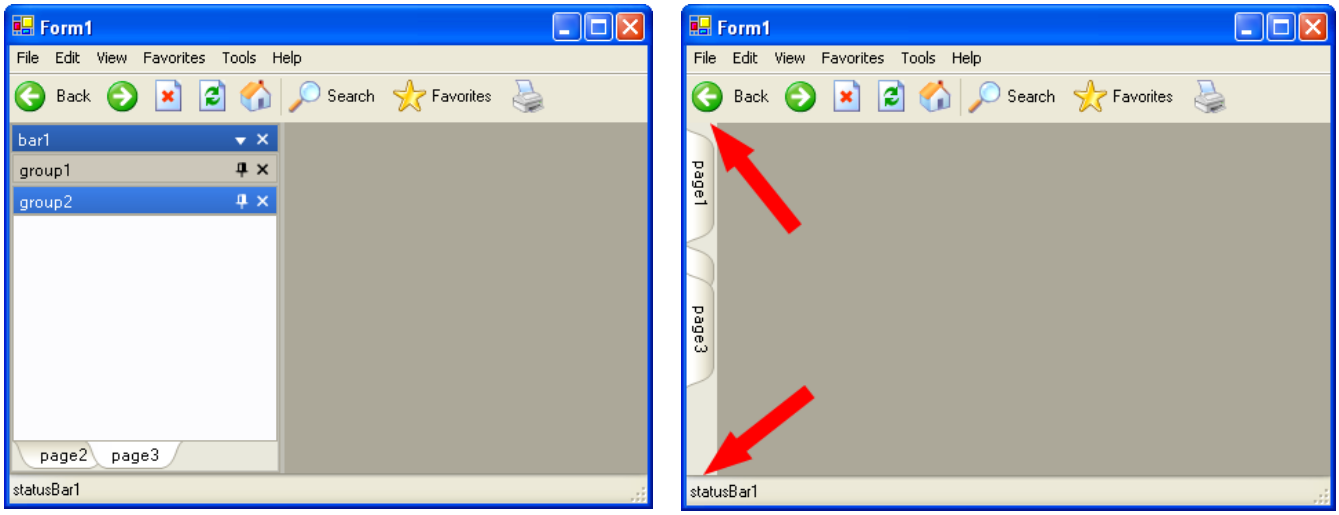



**Fig. B** - Creation of AutoHideBar control by handling of AutoHideChanged event for each group and setting the OuterControl property to the boundary control (in this case it's Toolbar control)




```
[Visual Basic]
Private Sub OnAutoHideChanged(ByVal sender As Object, ByVal ce As
   LidorSystems.Collector.CollectorEventArgs)
   ' Checking that auto-hidden object is Group
   If TypeOf ce.CollectorObject Is Group Then
      Dim currentGroup As Group = CType(ce.CollectorObject, Group)

      ' Checking the parent of Group control
      If TypeOf currentGroup.Parent Is AutoHideBar.AutoHidePanel Then
        Dim parentBar As AutoHideBar = CType(currentGroup.Parent,
AutoHideBar.AutoHidePanel).ParentBar

        ' Setting the OuterControl property to the boundary control, in this case
it is toolbar
        If Not (parentBar Is Nothing) Then
```

```
            parentBar.OuterControl = Me.toolbar1
         End If
      End If
   End If
End Sub

[C#]
private void OnAutoHideChanged(object sender,
LidorSystems.Collector.CollectorEventArgs ce)
{
   // Checking that auto-hidden object is Group
   if (ce.CollectorObject is Group)
   {
      Group currentGroup = (Group)ce.CollectorObject;


      // Checking the parent of Group control
      if (currentGroup.Parent is AutoHideBar.AutoHidePanel)
      {
         AutoHideBar parentBar =
(AutoHideBar.AutoHidePanel)currentGroup.Parent).ParentBar;


         // Setting the OuterControl property to the boundary control, in this case
it is toolbar
         if (parentBar != null)
            parentBar.OuterControl = this.toolbar1;
      }
   }
}
```

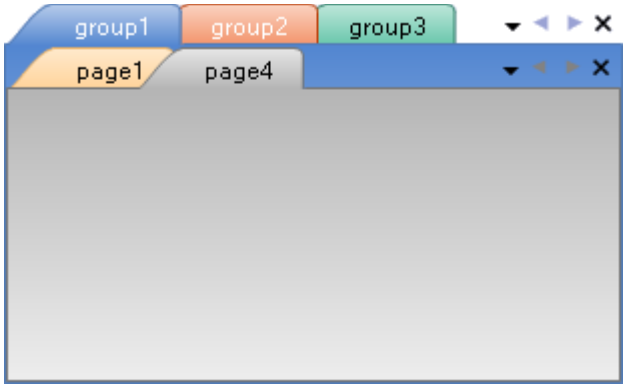
## Tabbed user interface

LidorSystems.Collector unites the possibilities of TabControl and Tabbed MDI in one place. To emulate TabControl, place the Group control on the Form and add some Page controls. Then hide the header of group with **Header** property value set to **False** and set the **BorderStyle** property to **None**. Make sure that group has **ControlState** property value set to **Static**. In this way is created control that looks like TabControl.
Creation of tabbed MDI is simple, just dock the group to one side of the Form and you will create tabbed user interface. You will notice that **ControlState** property will change to **Docked**.

Main difference between controls in static and docked state is that when control is docked you can drag&drop tabs from the tabstrip if you want to reorder documents, create floating windows or create new tabbed interface horizontally or vertically. This is not possible if the group is in static state.


If you set the **Appearance** property of **BarStyle** class to **Horizontal** you can create tabs in two levels, which will give you more freedom in categorization of documents. The following picture presents Bar control which orders groups horizontally with tabs, and in each group there are top oriented pages.

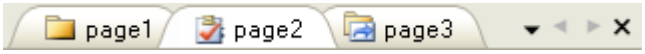
Set the **StyleFromParent** property to **False**, and change the **ColorScheme** for every group and page and you will create colorfull tabbed user interface.



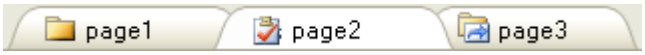
While working with tabs in some applications you may need to show disabled tabs. This can be done if you change the **Enabled** property value of Page control to **False** and fill the **PageStyle->TabStyle->DisabledStyle** with colors and fill style that you want.

There are three different sizes of tabs that you can use which are controlled with **TabMode** property:


**Compressed** - Only the active tab shows text and image; others show only image




**AutoSized** - The width of each tab is sized so that each tab can display text and image




**Justified** – The size of tabstrip is split amongst all tabs

## WinForms in tabular format

Table layout engine allows you to insert, arrange and operate with controls in design-time environment. If you are familiar working with tables, then you will easy notice the advantages and benefits this object can offer you. With this table object you can:

- Align controls in cells in nine positions
- Choose from three different size types of rows and columns (autosized, fixed and free) to easily arrange your controls
- Work with nested tables
- Merge and split cells
- Select, insert and delete rows, columns and table
- Operate with each cell, row or column in order to form, align and arrange the controls in the cells to achieve the best look for your application
- Context menu for easy manipulation with table
- Use **Table Properties** dialog box

To insert Table, right-click on empty Page control and from the context menu choose option to insert table with desired rows, columns, sizes and cell properties. The inserted table is in fact the **PageTable** object which will help you to place well-ordered controls. The borders of cells and entire table are visible only in design-time, while in runtime only content of cells will be visible. You can operate with each cell, row or column in order to form, align and arrange the controls in the cells to achieve the best look for your application. Most of the functions for this control are available through context menu with right-click over the table's cells or through **Table Properties** dialog box