



IntegralUI ListBox

v2.2 for .NET

Rich data presentation control

User Guide

for IntegralUI ListBox v2.2

Table of contents

Introduction	4
Architecture	5
Object Model	5
Event Model	6
Editor	7
Appearance	8
Working with styles	8
How to change the appearance of control border	8
Use of individual styles for items	9
How to create alternate look of items	10
Style serialization and reuse	12
Visual Styles	12
Themes	12
Behavior	13
Working with Item collection	13
Add/Remove operations	13
How to add existing item collection to the ListBox	14
How to edit the item text	14
Checked items	16
How to preserve checked items	16
Selected items	18
Multiple selection	18
Hover selection	18
How to preserve selection	18
Drag&Drop operations	19
Use of permissions	19
How to create a custom drag&drop operation	19
How to perform drag&drop for collection of items	22
How to perform drag&drop of an item content to other controls	29
Sorting	30
Use of predefined sorting method	30
Create custom sorting	31
Search	33
How to find an item by using specific criteria	33
How to get an item reference from mouse position	34
Keyword search	34
How to maintain scroll position	35

XML Encoding	36
XML Tags that are supported	36
Working with containers: <div> and <p> tags	39
Working with tag	40
Working with font style tags	41
Working with <a> tag	41
Working with tag	42
Working with <control> tag	43
Working with <table>, <tr> and <td> tags	44
Working with <style> tag	46
Using word wrap	48
Serialization	49
How to serialize the control content in Streams	49
How to serialize the control content in files	50
How to serialize the control content SQL database	52
Sample projects	54

Introduction

This guide will provide you with information on how to work with IntegralUI ListBox control and help you to successfully include it in your applications.

With IntegralUI ListBox you can present your data in very customizable way. This control allows you to include custom content in every item. By using XML tags you can create rich content.

Here are some of the main features:

- Highly customizable appearance
- Rich content: Text, Images, Hyperlinks, Controls, CheckBox, Flags; can be included in every item
- Arrange item content in custom layouts
- Advanced Drag&Drop operations
- Fast loading along with custom progress presentation
- XML encoding & serialization
- Theme support

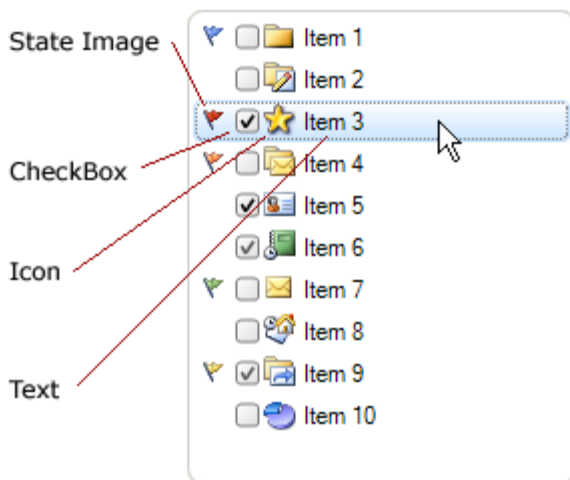
Architecture

IntegralUI ListBox control is part of IntegralUI family of products, and as such uses common infrastructure shared among other controls. The ListBox class is inherited from ListBase class which is a parent class for all list controls in IntegralUI class library. Because of this, you will find many similarities between other controls like ListView, ListBox and TreeListView, which all belongs to the IntegralUI Lists class library.

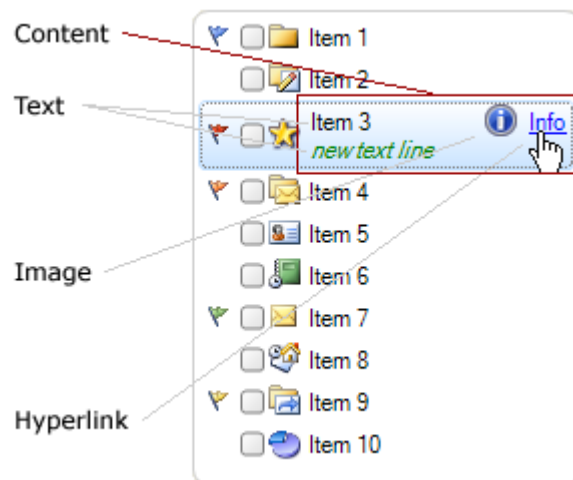
Object model



To fill the ListBox control with data you need to use **ListBoxItemCollection** which holds a collection of items. Every item can contain **StateImage**, **CheckBox**, **Icon**, **Text** and **Content**. If you want to present not just Text in the item, then the best is to use the **Content**. By using XML tags you can create rich content for every item including Text, Images, Hyperlinks, Controls, CheckBox, Flags, arranging it in custom layouts by using Tables and paragraphs.



Normal mode



Using XML Encoding

In order to create custom look you can use many color and format styles for ListBox control and for every item individually. Additionally, in further customization the styles can be changed for every part of the item content by using special XML tags.



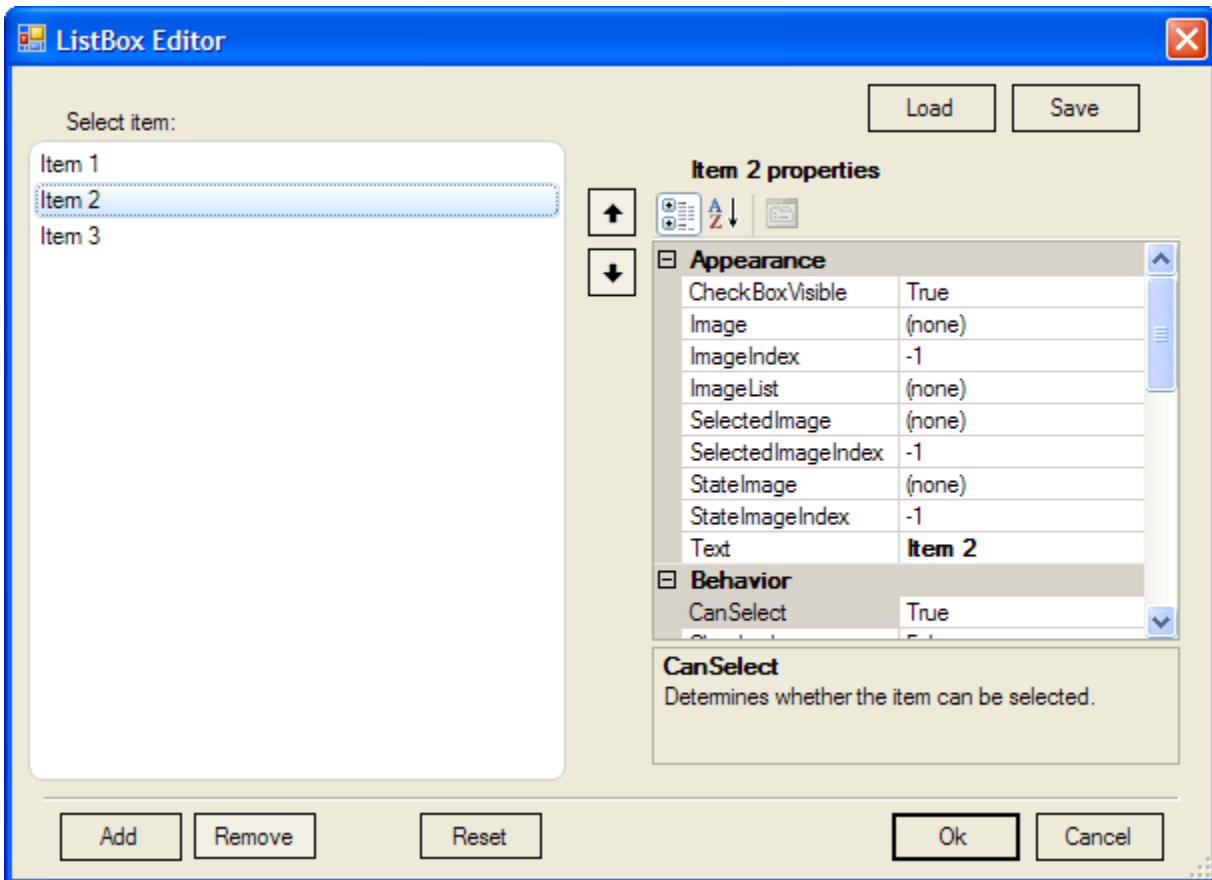
Event model

There are many events which can help you to determine the current state of ListBox control. Here is a complete list of events:

- AfterCheck Occurs after the item's check box is checked
- AfterLabelEdit Occurs after the item text is edited
- AfterSelect Occurs after the item is selected
- BeforeCheck Occurs before the item's check box is checked
- BeforeLabelEdit Occurs before the item text is edited
- BeforeSelect Occurs before the item is selected
- FocusedItemChanged Occurs after the focused item is changed
- FocusedItemChanging Occurs before the focused item is changed
- ItemAdded Occurs after new item is added
- ItemAdding Occurs before new item is added
- ItemDrag Occurs when the user begins dragging an item
- ItemMouseHover Occurs when the mouse cursor hovers for some time over an item space
- ItemObjectClicked Occurs after the user clicks a custom object (placed with XML encoding) in the control space
- ItemObjectClicking Occurs before the user clicks a custom object (placed with XML encoding) in the control space
- ItemRemoved Occurs after item is removed from collection
- ItemRemoving Occurs before item is removed from collection
- ScrollPosChanged Occurs when position of scrollbar has changed
- SelectionModeChanged Occurs when selection type changes

Editor

There are two ways to create items, during design-time or programmatically. In design-time you can use the ListBox Editor by right-clicking on the ListBox control. The form will appear in which you can add, remove, change the item order in collection or set the properties for each item.



In order to create items programmatically, you can use the following code:

```
[VB]
' Assuming that ListBox control exist
Dim item As New LidorSystems.IntegralUI.Lists.ListBoxItem("Item")
Me.listBox1.Items.Add(item)
Me.listBox1.UpdateLayout()

[C#]
// Assuming that ListBox control exist
LidorSystems.IntegralUI.Lists.ListBoxItem item = new
LidorSystems.IntegralUI.Lists.ListBoxItem("Item");
this.listBox1.Items.Add(item);
this.listBox1.UpdateLayout();
```

Appearance

Working with styles

The IntegralUI ListBox is very customizable control. You can customize every part of the control, starting from the background, border, items etc. This is done by numerous styles with which you can set colors, fonts, alignment, rendering mode. Here is the list of color and format styles:

- **CheckBoxStyle** - The drawing style of the item's check box
- **ColorStyle** - The drawing style of the control
- **FormatStyle** - The format style of the control
- **ScrollBarStyle** - The drawing style of horizontal and vertical scrollbar
- **ToolTipStyle** - The drawing style of item's tooltip

To simplify control over item's appearance, there are general styles which are used for all items:

- **DisabledItemStyle** - The drawing style of item when it's disabled
- **FocusedItemStyle** - The drawing style of item with input focus
- **HoverItemStyle** - The drawing style of item when mouse hovers over it
- **ItemFormatStyle** - Style by which the item content is formatted
- **NormalItemStyle** - The default drawing style of the item
- **SelectedItemStyle** - The drawing style of item when it's selected

Furthermore, every item have their own set of the above styles with witch you can customize their appearance individually. These styles are:

- **DisabledStyle** - The drawing style of item when it's disabled
- **FocusedStyle** - The drawing style of item with input focus
- **HoverStyle** - The drawing style of item when mouse hovers over it
- **FormatStyle** - Style by which the item content is formatted
- **NormalStyle** - The default drawing style of the item
- **SelectedStyle** - The drawing style of item when it's selected

How to change the appearance of control border

In some cases you may want to change the border of the ListBox to appear differently then the standard rectangular form. To achieve this goal several properties of the control FormatStyle needs to be changed. Here is a list of properties which controls a different part of the border:

- BorderCornerRadius** – holds the value by which the border corner is rounded
- BorderCornerShape** – responsible for changing the shape of every border corner
- BorderLineStyle** – determines the thickness of border line
- BorderVisibility** – determines which side of the border is visible

These properties also exist in format style of every item.

Here is an example where you can see how border can be customized:

```
[VB]
Imports LidorSystems.IntegralUI.Style
. . .

' Changing the border of the ListBox control
Me.listBox1.FormatStyle.BorderCornerRadius = 15
Me.listBox1.FormatStyle.BorderCornerShape.BottomLeft = CornerShape.Squared
Me.listBox1.FormatStyle.BorderCornerShape.BottomRight = CornerShape.Chamfered
```

```

Me.listBox1.FormatStyle.BorderCornerShape.TopRight = CornerShape.Squared
Me.listBox1.FormatStyle.BorderLineStyle = LineStyle.Double

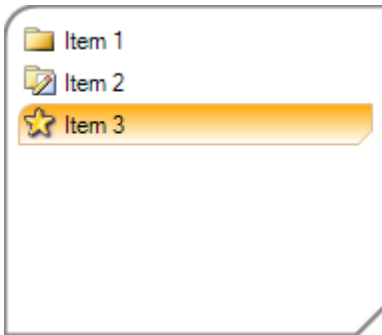
' Changing the border of the items
Me.listBox1.ItemFormatStyle.BorderCornerRadius = 7
Me.listBox1.ItemFormatStyle.BorderCornerShape.BottomLeft = CornerShape.Squared
Me.listBox1.ItemFormatStyle.BorderCornerShape.BottomRight =
CornerShape.Chamfered
Me.listBox1.ItemFormatStyle.BorderCornerShape.TopRight = CornerShape.Squared
Me.listBox1.ItemFormatStyle.Padding = New Padding(3, 2, 3, 2)

[C#]
using LidorSystems.IntegralUI.Style;
. . .

// Changing the border of the ListBox control
this.listBox1.FormatStyle.BorderCornerRadius = 15;
this.listBox1.FormatStyle.BorderCornerShape.BottomLeft = CornerShape.Squared;
this.listBox1.FormatStyle.BorderCornerShape.BottomRight = CornerShape.Chamfered;
this.listBox1.FormatStyle.BorderCornerShape.TopRight = CornerShape.Squared;
this.listBox1.FormatStyle.BorderLineStyle = LineStyle.Double;

// Changing the border of the items
this.listBox1.ItemFormatStyle.BorderCornerRadius = 7;
this.listBox1.ItemFormatStyle.BorderCornerShape.BottomLeft =
CornerShape.Squared;
this.listBox1.ItemFormatStyle.BorderCornerShape.BottomRight =
CornerShape.Chamfered;
this.listBox1.ItemFormatStyle.BorderCornerShape.TopRight = CornerShape.Squared;
this.listBox1.ItemFormatStyle.Padding = new Padding(3, 2, 3, 2);

```



Use of individual styles for items

By default the appearance of items is controlled by set of styles from their parent ListBox control.

If you want to have separate look for a specific item, you can customize it from their individual styles. Before changing any of these styles, the StyleFromParent property for this item must be set to False. In the following example we will change the look of the item in its normal and hovered state:

```

[VB]
' Set the StyleFromParent to False, so that
' a specific style changes be applied
item.StyleFromParent = False

' Change the look of the item, when it is in normal state
item.NormalStyle.BackColor = Color.LightGreen
item.NormalStyle.BorderColor = Color.LimeGreen

' Change the look of the item, when it is in hovered state
item HoverStyle.BackColor = Color.LightSalmon
item HoverStyle.BorderColor = Color.Salmon
item HoverStyle.FillStyle = FillStyle.Vertical

' Repairing the control
Me.listBox1.Invalidate()

[C#]
// Set the StyleFromParent to False, so that
// a specific style changes be applied
item.StyleFromParent = false;

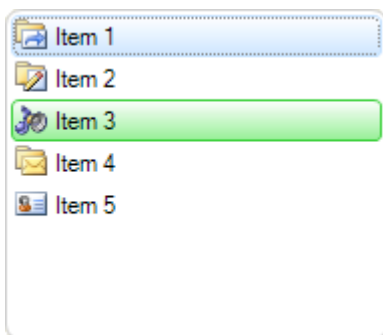
// Change the look of the item, when it is in normal state
item.NormalStyle.BackColor = Color.LightGreen;
item.NormalStyle.BorderColor = Color.LimeGreen;

// Change the look of the item, when it is in hovered state
item HoverStyle.BackColor = Color.LightSalmon;
item HoverStyle.BorderColor = Color.Salmon;
item HoverStyle.FillStyle = FillStyle.Vertical;

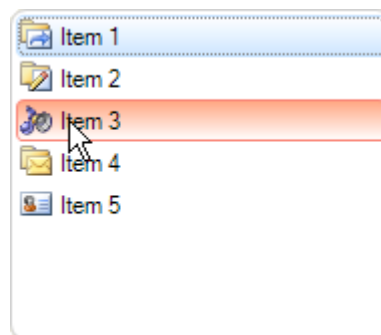
// Repairing the control
this.listBox1.Invalidate();

```

In the code above the item is the third item in the collection. Here is the result:



Changes in normal state



Changes in hover state

How to create alternate look of items

You can create standalone styles which in appropriate conditions can be applied to specific items. For example, every item with an even index can have one appearance and every item with odd index can have another appearance. But instead of changing the colors for every item, you need only to create two color styles, one style for the even rows and other for the odd rows.

Here is a sample code that creates item list with alternate look:

```
[VB]
Imports LidorSystems.IntegralUI.Lists
Imports LidorSystems.IntegralUI.Lists.Style

. . .

' Create the even color style
Dim evenItemStyle As New ListItemColorStyle()
evenItemStyle.BackColor = Color.WhiteSmoke
evenItemStyle.FillStyle = FillStyle.Horizontal

' Create the odd color style
Dim oddItemStyle As New ListItemColorStyle()
oddItemStyle.BackColor = Color.Gainsboro
oddItemStyle.FillStyle = FillStyle.Horizontal

' Create an item list and apply the styles
Dim item As ListBoxItem = Nothing
For i As Integer = 0 To 9
    item = New ListBoxItem("Item " & i.ToString())

    ' When you use individual styles for an item
    ' the StyleFromParent needs to be set to False
    item.StyleFromParent = False
    If i Mod 2 = 0 Then
        item.NormalStyle = evenItemStyle
    Else
        item.NormalStyle = oddItemStyle
    End If

    Me.listBox1.Items.Add(item)
Next

[C#]
using LidorSystems.IntegralUI.Lists;
using LidorSystems.IntegralUI.Lists.Style;

. . .

// Create the even color style
ListItemColorStyle evenItemStyle = new ListItemColorStyle();
evenItemStyle.BackColor = Color.WhiteSmoke;
evenItemStyle.FillStyle = FillStyle.Horizontal;

// Create the odd color style
ListItemColorStyle oddItemStyle = new ListItemColorStyle();
oddItemStyle.BackColor = Color.Gainsboro;
oddItemStyle.FillStyle = FillStyle.Horizontal;

// Create an item list and apply the styles
ListBoxItem item = null;
for (int i = 0; i < 10; i++)
{
    item = new ListBoxItem("Item " + i.ToString());

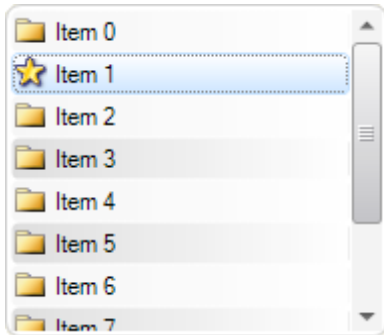
    // When you use individual styles for an item
```

```

// the StyleFromParent needs to be set to False
item.StyleFromParent = false;
if (i % 2 == 0)
    item.NormalStyle = evenItemStyle;
else
    item.NormalStyle = oddItemStyle;

this.listBox1.Items.Add(item);
}

```



Style serialization and reuse

If you want to create some predefined color schemes and apply it to the control at specific conditions, one way to do that is by serializing the control layout. You only would need to set the color and format styles and then save the layout in external xml file or database. After that, you only need to load the file which corresponds to the target color scheme.

You can read more on how serialization is done at the end of this document in Serialization section.

Visual Styles

There are three predefined visual styles:

- **Classic** - Used to render controls in Windows Classic control style
- **XP** - Used to render controls in Windows XP control style
- **Vista** - Used to render controls in Windows Vista control style

By setting the VisualStyle property to some of the above options, different appearance is applied to the control.

Themes

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**. Depending of the current **VisualStyle** a predefined color scheme is used which corresponds to the current operating system color scheme.

Note For creation of more advanced appearance, you can read in XML encoding section.

Behavior

Working with Item collection

Add/Remove operations

The items displayed in the ListBox control are stored in Items property. This is an object from ListBoxItemCollection class, which has several methods and events you can use to add, remove, clear or make any other operation that will change the collection. Here is a list of methods that you can use:

- Add – adds a new item at the end of the collection
- AddRange – add a set of items to the end of the collection
- Clear – empties the list
- Remove – deletes the item from the collection
- RemoveAt – deletes the item that is located at specified location in the collection

Here is an example how to add a new item programmatically to the collection:

```
[VB]
Dim item As New ListBoxItem("New item")
Me.listBox1.Items.Add(item)

[C#]
ListBoxItem item = new ListBoxItem("New item");
this.listBox1.Items.Add(item);
```

To insert this item at a specified position in the collection, use the Insert method:

```
[VB]
' Adds the item at sixth position in the collection
Me.listBox1.Items.Insert(5, item)

[C#]
// Adds the item at sixth position in the collection
this.listBox1.Items.Insert(5, item);
```

When you add a new item to the ListBox control, this process is accompanied with two events:

- ItemAdding – it is fired before item is added and can cancel the add operation
- ItemAdded – it is fired after the item is added

To remove an item from the collection, two methods can be used:

```
[VB]
' Removes the item from the collection
Me.listBox1.Items.Remove(item)

' Removes the item located at the specified location from the collection
Me.listBox1.Items.RemoveAt(5)

[C#]
// Removes the item from the collection
this.listBox1.Items.Remove(item);

// Removes the item located at the specified location from the collection
this.listBox1.Items.RemoveAt(5);
```

If you want to remove all items, use the Clear method

```
[VB]
Me.listBox1.Items.Clear();

[C#]
this.listBox1.Items.Clear();
```

How to add existing item collection

If you have some predefined set of items, and you want this set as a whole to be added to the ListBox control, you can use AddRange method:

```
[VB]
Imports LidorSystems.IntegralUI.Lists

. . .

Dim items As ListBoxItem() = New ListBoxItem(6) {}
For i As Integer = 0 To 6
    items(i) = New ListBoxItem("Item " & i.ToString())
Next

Me.listBox1.Items.AddRange(items)

[C#]
using LidorSystems.IntegralUI.Lists;

. . .

ListBoxItem[] items = new ListBoxItem[7];
for (int i = 0; i < 7; i++)
    items[i] = new ListBoxItem("Item " + i.ToString());

this.listBox1.Items.AddRange(items);
```

How to edit the item text

During runtime you can allow to users to edit the item text and change it. This can be done by setting the **LabelEdit** property to **True**. After that whenever the user clicks on the item and releases the mouse button, a TextBox will appear in which the item text can be edited.

If you want to start editing process in other way, you need to use **BeginEdit** and **EndEdit** methods. In the following example we will show how to programmatically start editing process:

```
[VB]
If Me.ListBox1.SelectedItem IsNot Nothing Then
    Me.ListBox1.LabelEdit = True
    Me.ListBox1.SelectedItem.BeginEdit()
End If

[C#]
```

```

if (this.listBox1.SelectedItem != null)
{
    this.listBox1.LabelEdit = true;
    this.listBox1.SelectedItem.BeginEdit();
}

```

The edit process is accompanied by two events:

- **BeforeLabelEdit** – it is fired before the TextEditor is shown and editing begins
- **AfterLabelEdit** – it is fired when TextEditor closes and just before the new text is applied to the item Text property

In order to cancel the current edit process, you can simply press the ESC key or by handle the AfterLabelEdit event. In the following example the edit is allowed only for labels which don't contain some characters:

```

[VB]
Imports LidorSystems.IntegralUI.Lists

. . .

Private Sub listBox1_AfterLabelEdit(ByVal sender As Object, ByVal e As
LidorSystems.IntegralUI.ObjectEditEventArgs)
    If TypeOf e.[Object] Is ListBoxItem Then
        Dim item As ListBoxItem = DirectCast(e.[Object], ListBoxItem)

        If e.Label IsNot Nothing Then
            If e.Label.Length > 0 Then
                If e.Label.IndexOfAny(New Char() {"@","c", ".", "c", ",","c", "!"c})) >= 0 Then
                    ' Cancel the label edit action, inform the user, and
                    ' place the node in edit mode again.
                    e.Cancel = True

                    MessageBox.Show("Invalid item label." & vbLf & "The invalid
characters are: '@','.',',','!',", "Item Label Edit")
                    item.BeginEdit()
                End If
            Else
                ' Cancel the label edit action, inform the user, and
                ' place the node in edit mode again.
                e.Cancel = True

                MessageBox.Show("Invalid item label." & vbLf & "The label cannot be
blank", "Item Label Edit")
                item.BeginEdit()
            End If
        End If
    End If
End Sub

[C#]
using LidorSystems.IntegralUI.Lists;

. . .

private void listBox1_AfterLabelEdit(object sender,
LidorSystems.IntegralUI.ObjectEditEventArgs e)

```

```

{
    if (e.Object is ListBoxItem)
    {
        ListBoxItem item = (ListBoxItem)e.Object;

        if (e.Label != null)
        {
            if (e.Label.Length > 0)
            {
                if (e.Label.IndexOfAny(new char[] { '@', '.', ',', '!' }) >= 0)
                {
                    // Cancel the label edit action, inform the user, and
                    // place the node in edit mode again.
                    e.Cancel = true;

                    MessageBox.Show("Invalid item label.\n" + "The invalid
characters are: '@', '.', ',', '!', "Item Label Edit");
                    item.BeginEdit();
                }
            }
            else
            {
                // Cancel the label edit action, inform the user, and
                // place the node in edit mode again.
                e.Cancel = true;

                MessageBox.Show("Invalid item label.\nThe label cannot be blank",
"Item Label Edit");
                item.BeginEdit();
            }
        }
    }
}
}

```

Checked items

In order to ListBox act as Checked ListBox control, the **CheckBoxes** property must be set to **True**. Every item has a check box shown to the left side of the item space. Their visibility is controlled by **CheckBoxVisible** property. In this way, you can decide which items will have checkbox. This is useful for example when you want some item to behave like a header for other items.

The check box can display two or three state values. By default the two-state (checked and unchecked) behavior is active. If you want to have three-state behavior, at first you need to set the **CheckMode** to **ThreeState** value. Whenever user clicks inside the check box space, it will cycle the state of the check box through Unchecked, Indeterminate and Checked values.

In some cases you will also want to change the check box state by item selection. In order to do that, set the **AllowSelectionCheck** property to **True**.

The set of checked items (only those with **CheckState** set to **Checked**) is stored in **CheckedItems** collection. This collection is very useful when you want to examine only items that are currently checked.

How to preserve checked items

In some cases you may want to preserve the collection of checked items unchanged. For example, there may be custom operation which will indirectly change the CheckState of the items, and in this way the CheckedItems collection. In order to prevent this from happening, we have added a **PreserveCheckState** property. With it you can preserve the current collection, run your custom operation without side effects, and then set back this property to its original value. For example:

```
[VB]
Private Sub button1_Click(ByVal sender As Object, ByVal e As EventArgs)
    Me.listBox1.PreserveCheckState = True
    Me.listBox1.SelectedItem = Me.listBox1.Items(1)
    Me.listBox1.PreserveCheckState = False
End Sub

Private Sub listBox1_AfterSelect(ByVal sender As Object, ByVal e As
LidorSystems.IntegralUI.ObjectEventArgs)
    If TypeOf e.[Object] Is LidorSystems.IntegralUI.Lists.ListBoxItem Then
        Dim item As LidorSystems.IntegralUI.Lists.ListBoxItem = DirectCast(e.
[Object], LidorSystems.IntegralUI.Lists.ListBoxItem)

        Me.listBox1.CheckedItems.Clear()
        item.Checked = True
    End If
End Sub

[C#]
private void button1_Click(object sender, EventArgs e)
{
    this.listBox1.PreserveCheckState = true;
    this.listBox1.SelectedItem = this.listBox1.Items[1];
    this.listBox1.PreserveCheckState = false;
}

private void listBox1_AfterSelect(object sender,
LidorSystems.IntegralUI.ObjectEventArgs e)
{
    if (e.Object is LidorSystems.IntegralUI.Lists.ListBoxItem)
    {
        LidorSystems.IntegralUI.Lists.ListBoxItem item =
(LidorSystems.IntegralUI.Lists.ListBoxItem)e.Object;

        this.listBox1.CheckedItems.Clear();
        item.Checked = true;
    }
}
```

In this example we have a ListBox with some items in it. Whenever a button is clicked the second item is selected and checked. But along with this by calling the Clear method, we try to remove all checked items from CheckedItems collection. Because we have set **PreserveCheckState** to **True**, this collection will remain intact.

Selected items

Multiple selection

The control supports four different modes for item selection. They are controlled from the `SelectionMode` property, which can have following values:

- None – there is no selection
- One – only one item can be selected
- MultiSimple – selection is performed with single mouse click
- MultiExtended – selection is performed with use of CTRL or SHIFT keys

In order to have multiple item selection, the `SelectionMode` must be set either to `MultiSimple` or `MultiExtended` value.

Whenever there is a selection, items that are currently selected are stored in `SelectedItems` collection. This collection is very useful when you want to examine only items that are currently selected.

Hover selection

In some cases you may want to allow items to be selected while mouse cursor hovers over them. This functionality is built-in the code, and in order to allow it, just set the **HoverSelection** property to **True**.

How to preserve selection

In some cases you may want to preserve the collection of selected items unchanged. For example, there may be custom operation which will indirectly change the `Selected` state of the item, and in this way the `SelectedItems` collection. In order to prevent this from happening, we have added a **PreserveSelection** property. With it you can preserve the current collection, run your custom operation without side effects, and then set back this property to its original value. For example:

```
[VB]
Private Sub button1_Click(ByVal sender As Object, ByVal e As EventArgs)
    Me.listBox1.PreserveSelection = True
    Me.listBox1.SelectedItems.Clear()
    Me.listBox1.SelectedItem = Me.listBox1.Items(1)
    Me.listBox1.PreserveSelection = False
End Sub

[C#]
private void button1_Click(object sender, EventArgs e)
{
    this.listBox1.PreserveSelection = true;
    this.listBox1.SelectedItems.Clear();
    this.listBox1.SelectedItem = this.listBox1.Items[1];
    this.listBox1.PreserveSelection = false;
}
```

In this example we have a `ListBox` with some items in it. Whenever a button is clicked the second item is selected. But along with this by calling the `Clear` method, we try to remove all selected items from `SelectedItems` collection. Because we have set **PreserveSelection** to **True**, this collection will remain intact. This only works in multiple selection mode.

Drag&Drop operations

The IntegralUI ListBox control has built-in support for standard drag&drop operations like: drag and drop of single or multiple items, copy/move the item/s from one ListBox to other controls etc. In the following sections we will give you information on how you can use various permissions during drag&drop operations, and how to extend this by creating your own custom operation.

Use of permissions

Here is a brief description on various properties that are used during drag&drop:

- **AllowDrag** – Gives permission to item/s to be dragged
- **AllowDrop** – Gives permission to the control to accept item/s during drag&drop
- **DragDropMode** – Determines the type of drag&drop operation
- **DragDropReorder** – Determines whether a reordering of items is allowed during drag&drop
- **ShowDropMarker** – Determines whether the marker is shown that represents the current drop position

By default drag&drop is disabled. In order to start dragging of item/s, the **AllowDrag** property must be set to **True**. However, in order for dragged items to be dropped, the **AllowDrop** property must also be set to **True**. We have deliberately separated the control over drag&drop operation in two states, because in some cases you may want to have one ListBox control from which items can only be dragged, and other ListBox control to which items can only be dropped.

By default there is a predefined drag&drop operation which handles all of standard cases. In order to create your own operation, the **DragDropMode** must be set to **Custom**. This case is described in detail in the next section.

If you don't want to appear the drop marker during drag&drop, the best is to disable it from the **ShowDropMarker** property. In other case, this marker will be shown. You can change the color of this marker from the ColorStyle of the ListBox, by changing the **ItemPosColor** property.

How to create a custom drag&drop operation

If the default drag&drop operation doesn't give you the solution you want, you can always create your own custom operation.

In order to do that, you need to do the following:

1. At first, you need to set the **DragDropMode** to **Custom**
2. Handle the **ItemDrag** event and start the drag&drop operation
3. Handle the **DragOver** and **DragDrop** events
4. Optionally, handle other drag&drop events, like: **DragEnter**, **DragLeave**, **QueryContinueDrag**, etc.

Here is some code that you can use:

```
[VB]
Imports LidorSystems.IntegralUI.Collections
Imports LidorSystems.IntegralUI.Lists
Imports LidorSystems.IntegralUI.Lists.Collections
```

```

. . .

Private Sub listBox1_ItemDrag(ByVal sender As Object, ByVal e As
ItemDragEventArgs)
    If e.Button = MouseButtons.Left Then
        Me.listBox1.DoDragDrop(e.Item, DragDropEffects.All)
    End If
End Sub

Private Sub listBox1_DragOver(ByVal sender As Object, ByVal e As DragEventArgs)
    If e.Data.GetDataPresent(GetType(ListBoxItem)) Then
        ' Depending od the control key pressed change the drag effect
        If (e.KeyState And 8) = 8 AndAlso (e.AllowedEffect And
DragDropEffects.Copy) = DragDropEffects.Copy Then
            e.Effect = DragDropEffects.Copy
        Else
            e.Effect = DragDropEffects.Move
        End If
    Else
        e.Effect = DragDropEffects.None
    End If
End Sub

Private Sub listBox1_DragDrop(ByVal sender As Object, ByVal e As DragEventArgs)
    Dim lb As LidorSystems.IntegralUI.Lists.ListBox = DirectCast(sender,
LidorSystems.IntegralUI.Lists.ListBox)

    ' Get the current mouse position
    Dim mousePos As Point = lb.PointToClient(New Point(e.X, e.Y))

    If e.Data.GetDataPresent(GetType(ListBoxItem)) Then
        ' Get the dragged item
        Dim item As ListBoxItem = DirectCast(e.Data.GetData(GetType(ListBoxItem)),
ListBoxItem)

        ' Get the target item (the one that is currently hovered)
        Dim targetItem As ListBoxItem = Me.listBox1.GetHoverItem()

        ' In Copy operation, create a clone item and then add it to the target
        If e.Effect = DragDropEffects.Copy Then
            item = DirectCast(item.Clone(), ListBoxItem)
        End If

        ' Suspend the listbox layout to increase performance
        Me.listBox1.SuspendUpdate()

        ' If there is no target item, then add the dragged item at the end
        If targetItem Is Nothing Then
            Me.listBox1.Items.SetChildIndex(item, Me.listBox1.Items.Count - 1)
        Else
            ' Get the index of the dragged item in target ListBox control
            Dim newIndex As Integer = Me.listBox1.GetDropPos(targetItem, mousePos)
            If newIndex >= 0 Then
                Me.listBox1.Items.SetChildIndex(item, newIndex)
            Else
                Me.listBox1.Items.SetChildIndex(item, Me.listBox1.Items.Count - 1)
            End If
        End If
    End If
End Sub

```

```

        ' Resume the listbox layout and update the control
        Me.listBox1.ResumeUpdate()
    End If
End Sub

Private Sub listBox1_QueryContinueDrag(ByVal sender As Object, ByVal e As
QueryContinueDragEventArgs)
    Dim mousePos As Point = Me.listBox1.PointToClient(Control.MousePosition)

    ' Cancel the drag operation when the mouse cursor leaves the ListBox space
    If Not Me.listBox1.ClientRectangle.Contains(mousePos) Then
        e.Action = DragAction.Cancel
    End If
End Sub

[C#]
using LidorSystems.IntegralUI.Collections;
using LidorSystems.IntegralUI.Lists;
using LidorSystems.IntegralUI.Lists.Collections;

. . .

private void listBox1_ItemDrag(object sender, ItemDragEventArgs e)
{
    if (e.Button == MouseButtons.Left)
        this.listBox1.DoDragDrop(e.Item, DragDropEffects.All);
}

private void listBox1_DragOver(object sender, DragEventArgs e)
{
    if (e.Data.GetDataPresent(typeof(ListBoxItem)))
    {
        // Depending od the control key pressed change the drag effect
        if ((e.KeyState & 8) == 8 && (e.AllowedEffect &
DragDropEffects.Copy) == DragDropEffects.Copy)
            e.Effect = DragDropEffects.Copy;
        else
            e.Effect = DragDropEffects.Move;
    }
    else
        e.Effect = DragDropEffects.None;
}

private void listBox1_DragDrop(object sender, DragEventArgs e)
{
    LidorSystems.IntegralUI.Lists.ListBox lb =
(LidorSystems.IntegralUI.Lists.ListBox)sender;

    // Get the current mouse position
    Point mousePos = lb.PointToClient(new Point(e.X, e.Y));

    if (e.Data.GetDataPresent(typeof(ListBoxItem)))
    {
        // Get the dragged item
        ListBoxItem item =
(ListBoxItem)e.Data.GetData(typeof(ListBoxItem));

        // Get the target item (the one that is currently hovered)
        ListBoxItem targetItem = this.listBox1.GetHoverItem();

```



```

Private Sub listBox1_ItemDrag(ByVal sender As Object, ByVal e As
ItemDragEventArgs)
    If e.Button = MouseButtons.Left Then

        Dim itemCollection As New ObjectCollection()
        For Each item As ListBoxItem In Me.listBox1.CheckedItems
            itemCollection.Add(item)
        Next

        Me.listBox1.DoDragDrop(itemCollection, DragDropEffects.All)
    End If
End Sub

Private Sub listBox_DragOver(ByVal sender As Object, ByVal e As DragEventArgs)
    If e.Data.GetDataPresent(GetType(ObjectCollection)) Then
        ' Depending od the control key pressed change the drag effect
        If (e.KeyState And 8) = 8 AndAlso (e.AllowedEffect And
DragDropEffects.Copy) = DragDropEffects.Copy Then
            e.Effect = DragDropEffects.Copy
        Else
            e.Effect = DragDropEffects.Move
        End If
    Else
        e.Effect = DragDropEffects.None
    End If
End Sub

Private Sub listBox_DragDrop(ByVal sender As Object, ByVal e As DragEventArgs)
    Dim lb As LidorSystems.IntegralUI.Lists.ListBox = DirectCast(sender,
LidorSystems.IntegralUI.Lists.ListBox)

    ' Get the current mouse position
    Dim mousePos As Point = lb.PointToClient(New Point(e.X, e.Y))

    Dim targetList As LidorSystems.IntegralUI.Lists.ListBox = Nothing
    Dim targetCtrl As Control = Me.listBox1.GetTargetControl()
    If TypeOf targetCtrl Is LidorSystems.IntegralUI.Lists.ListBox Then
        targetList = DirectCast(targetCtrl, LidorSystems.IntegralUI.Lists.ListBox)
    ElseIf TypeOf targetCtrl Is LidorSystems.IntegralUI.Containers.InnerPanel
Then
        targetList = DirectCast(targetCtrl.Parent,
LidorSystems.IntegralUI.Lists.ListBox)
    End If

    If e.Data.GetDataPresent(GetType(ObjectCollection)) Then
        If targetList IsNot Nothing Then
            ' Get the dragged item collection
            Dim objects As ObjectCollection =
DirectCast(e.Data.GetData(GetType(ObjectCollection)), ObjectCollection)
            Dim itemCollection As New ListBoxItemCollection()
            For Each obj As Object In objects
                If TypeOf obj Is ListBoxItem Then
                    itemCollection.Add(obj)
                End If
            Next

            ' Get the target item (the one that is currently hovered)
            Dim targetItem As ListBoxItem = targetList.GetHoverItem()

```

```

If Me.listBox1 <> targetList Then
    ' Suspend the listbox layout to increase performance
    Me.listBox1.SuspendUpdate()
    targetList.SuspendUpdate()

    ProcessTargetDragDrop(e, targetList, targetItem, itemCollection)

    ' Resume the listbox layout and update the control
    Me.listBox1.ResumeUpdate()
    targetList.ResumeUpdate()
Else
    ' Suspend the listbox layout to increase performance
    targetList.SuspendUpdate()

    ProcessInsideDragDrop(e, targetList, targetItem, itemCollection)

    ' Resume the listbox layout and update the control
    targetList.ResumeUpdate()
End If
End If
End If
End Sub

Private Sub ProcessInsideDragDrop(ByVal e As DragEventArgs, ByVal targetList As
LidorSystems.IntegralUI.Lists.ListBox, ByVal targetItem As ListBoxItem, ByVal
itemCollection As ListBoxItemCollection)
    Dim mousePos As Point = targetList.PointToClient(New Point(e.X, e.Y))

    ' Get the collection at which nodes will be added
    If targetItem Is Nothing Then
        ' In Copy operation, create a clone item and then add it to the target
        If e.Effect = DragDropEffects.Copy Then
            For Each item As ListBoxItem In itemCollection
                targetList.Items.Add(item.Clone())
            Next

            ' In Move operation, it's best to use the while cycle, because when
moving existing node
            ' it will alter the source node collection which can cause errors.
            ' The best approach is the code bellow.
        ElseIf e.Effect = DragDropEffects.Move Then
            For Each item As ListBoxItem In itemCollection
                targetList.Items.SetChildIndex(item, targetList.Items.Count - 1)
            Next
        End If
    Else
        ' Get the index of the dropped item in target ListBox control
        Dim newIndex As Integer = targetList.GetDropPos(targetItem, mousePos)

        ' In Copy operation, create a clone item and then add it to the target
        If e.Effect = DragDropEffects.Copy Then
            For Each item As ListBoxItem In itemCollection
                If newIndex >= 0 Then
                    targetList.Items.Insert(newIndex, item.Clone())
                Else
                    targetList.Items.Add(item.Clone())
                End If
            Next
        End If
    End If
End Sub

```

```

        ' In Move operation, it's best to use the while cycle, because when
moving existing item
        ' it will alter the source item collection which can cause errors.
        ' The best approach is the item bellow.
ElseIf e.Effect = DragDropEffects.Move Then
    For Each item As ListBoxItem In itemCollection
        If newIndex >= 0 Then
            targetList.Items.SetChildIndex(item, newIndex)
        Else
            targetList.Items.SetChildIndex(item, targetList.Items.Count - 1)
        End If
    Next
End If
End If
End Sub

Private Sub ProcessTargetDragDrop(ByVal e As DragEventArgs, ByVal targetList As
LidorSystems.IntegralUI.Lists.ListBox, ByVal targetItem As ListBoxItem, ByVal
itemCollection As ListBoxItemCollection)
    Dim mousePos As Point = targetList.PointToClient(New Point(e.X, e.Y))

    ' Get the collection at which nodes will be added
    If targetItem Is Nothing Then
        ' In Copy operation, create a clone item and then add it to the target
        If e.Effect = DragDropEffects.Copy Then
            For Each item As ListBoxItem In itemCollection
                targetList.Items.Add(item.Clone())
            Next

            ' In Move operation, it's best to use the while cycle, because when
moving existing node
            ' it will alter the source node collection which can cause errors.
            ' The best approach is the code bellow.
            ElseIf e.Effect = DragDropEffects.Move Then
                For Each item As ListBoxItem In itemCollection
                    targetList.Items.Add(item)
                Next
            End If
        Else
            ' Get the index of the dropped item in target ListBox control
            Dim newIndex As Integer = targetList.GetDropPos(targetItem, mousePos)

            ' In Copy operation, create a clone item and then add it to the target
            If e.Effect = DragDropEffects.Copy Then
                For Each item As ListBoxItem In itemCollection
                    If newIndex >= 0 Then
                        targetList.Items.Insert(newIndex, item.Clone())
                    Else
                        targetList.Items.Add(item.Clone())
                    End If
                Next

                ' In Move operation, it's best to use the while cycle, because when moving
existing item
                ' it will alter the source item collection which can cause errors.
                ' The best approach is the item bellow.
                ElseIf e.Effect = DragDropEffects.Move Then
                    For Each item As ListBoxItem In itemCollection
                        If newIndex >= 0 Then

```

```

        targetList.Items.Insert(newIndex, item)
    Else
        targetList.Items.Add(item)
    End If
Next
End If
End If
End Sub

[C#]
using LidorSystems.IntegralUI.Collections;
using LidorSystems.IntegralUI.Lists;
using LidorSystems.IntegralUI.Lists.Collections;

. . .

private void listBox1_ItemDrag(object sender, ItemDragEventArgs e)
{
    if (e.Button == MouseButtons.Left)
    {
        ObjectCollection itemCollection = new ObjectCollection();
        foreach (ListBoxItem item in this.listBox1.CheckedItems)
            itemCollection.Add(item);

        this.listBox1.DoDragDrop(itemCollection, DragDropEffects.All);
    }
}

private void listBox_DragOver(object sender, DragEventArgs e)
{
    if (e.Data.GetDataPresent(typeof(ObjectCollection)))
    {
        // Depending od the control key pressed change the drag effect
        if ((e.KeyState & 8) == 8 && (e.AllowedEffect & DragDropEffects.Copy) ==
DragDropEffects.Copy)
            e.Effect = DragDropEffects.Copy;
        else
            e.Effect = DragDropEffects.Move;
    }
    else
        e.Effect = DragDropEffects.None;
}

private void listBox_DragDrop(object sender, DragEventArgs e)
{
    LidorSystems.IntegralUI.Lists.ListBox lb =
(LidorSystems.IntegralUI.Lists.ListBox)sender;

    // Get the current mouse position
    Point mousePos = lb.PointToClient(new Point(e.X, e.Y));

    LidorSystems.IntegralUI.Lists.ListBox targetList = null;
    Control targetCtrl = this.listBox1.GetTargetControl();
    if (targetCtrl is LidorSystems.IntegralUI.Lists.ListBox)
        targetList = (LidorSystems.IntegralUI.Lists.ListBox)targetCtrl;
    else if (targetCtrl is LidorSystems.IntegralUI.Containers.InnerPanel)
        targetList = (LidorSystems.IntegralUI.Lists.ListBox)targetCtrl.Parent;

    if (e.Data.GetDataPresent(typeof(ObjectCollection)))

```

```

    {
        if (targetList != null)
        {
            // Get the dragged item collection
            ObjectCollection objects =
(ObjectCollection)e.Data.GetData(typeof(ObjectCollection));
            ListBoxItemCollection itemCollection = new ListBoxItemCollection();
            foreach (object obj in objects)
            {
                if (obj is ListBoxItem)
                    itemCollection.Add(obj);
            }

            // Get the target item (the one that is currently hovered)
            ListBoxItem targetItem = targetList.GetHoverItem();

            if (this.listBox1 != targetList)
            {
                // Suspend the listbox layout to increase performance
                this.listBox1.SuspendUpdate();
                targetList.SuspendUpdate();

                ProcessTargetDragDrop(e, targetList, targetItem, itemCollection);

                // Resume the listbox layout and update the control
                this.listBox1.ResumeUpdate();
                targetList.ResumeUpdate();
            }
            else
            {
                // Suspend the listbox layout to increase performance
                targetList.SuspendUpdate();

                ProcessInsideDragDrop(e, targetList, targetItem, itemCollection);

                // Resume the listbox layout and update the control
                targetList.ResumeUpdate();
            }
        }
    }
}

private void ProcessInsideDragDrop(DragEventArgs e,
LidorSystems.IntegralUI.Lists.ListBox targetList, ListBoxItem targetItem,
ListBoxItemCollection itemCollection)
{
    Point mousePos = targetList.PointToClient(new Point(e.X, e.Y));

    // Get the collection at which nodes will be added
    if (targetItem == null)
    {
        // In Copy operation, create a clone item and then add it to the target
        if (e.Effect == DragDropEffects.Copy)
        {
            foreach (ListBoxItem item in itemCollection)
                targetList.Items.Add(item.Clone());
        }
        // In Move operation, it's best to use the while cycle, because when
        moving existing node
    }
}

```

```

// it will alter the source node collection which can cause errors.
// The best approach is the code bellow.
else if (e.Effect == DragDropEffects.Move)
{
    foreach (ListBoxItem item in itemCollection)
        targetList.Items.SetChildIndex(item, targetList.Items.Count - 1);
}
else
{
    // Get the index of the dropped item in target ListBox control
    int newIndex = targetList.GetDropPos(targetItem, mousePos);

    // In Copy operation, create a clone item and then add it to the target
    if (e.Effect == DragDropEffects.Copy)
    {
        foreach (ListBoxItem item in itemCollection)
        {
            if (newIndex >= 0)
                targetList.Items.Insert(newIndex, item.Clone());
            else
                targetList.Items.Add(item.Clone());
        }
    }
    // In Move operation, it's best to use the while cycle, because when
moving existing item
    // it will alter the source item collection which can cause errors.
    // The best approach is the item bellow.
    else if (e.Effect == DragDropEffects.Move)
    {
        foreach (ListBoxItem item in itemCollection)
        {
            if (newIndex >= 0)
                targetList.Items.SetChildIndex(item, newIndex);
            else
                targetList.Items.SetChildIndex(item, targetList.Items.Count-1);
        }
    }
}

private void ProcessTargetDragDrop(DragEventArgs e,
LidorSystems.IntegralUI.Lists.ListBox targetList, ListBoxItem targetItem,
ListBoxItemCollection itemCollection)
{
    Point mousePos = targetList.PointToClient(new Point(e.X, e.Y));

    // Get the collection at which nodes will be added
    if (targetItem == null)
    {
        // In Copy operation, create a clone item and then add it to the target
        if (e.Effect == DragDropEffects.Copy)
        {
            foreach (ListBoxItem item in itemCollection)
                targetList.Items.Add(item.Clone());
        }
        // In Move operation, it's best to use the while cycle, because when
moving existing node
        // it will alter the source node collection which can cause errors.

```

```

// The best approach is the code bellow.
else if (e.Effect == DragDropEffects.Move)
{
    foreach (ListBoxItem item in itemCollection)
        targetList.Items.Add(item);
}
else
{
    // Get the index of the dropped item in target ListBox control
    int newIndex = targetList.GetDropPos(targetItem, mousePos);

    // In Copy operation, create a clone item and then add it to the target
    if (e.Effect == DragDropEffects.Copy)
    {
        foreach (ListBoxItem item in itemCollection)
        {
            if (newIndex >= 0)
                targetList.Items.Insert(newIndex, item.Clone());
            else
                targetList.Items.Add(item.Clone());
        }
    }
    // In Move operation, it's best to use the while cycle, because when
    moving existing item
    // it will alter the source item collection which can cause errors.
    // The best approach is the item bellow.
    else if (e.Effect == DragDropEffects.Move)
    {
        foreach (ListBoxItem item in itemCollection)
        {
            if (newIndex >= 0)
                targetList.Items.Insert(newIndex, item);
            else
                targetList.Items.Add(item);
        }
    }
}
}
}

```

How to perform drag&drop of an item to other controls

Other controls can accept items during drag&drop operation, if their AllowDrop property is set to True. In the following example we will demonstrate how item dragged from the ListBox control can be dropped in TextBox control:

```

[VB]
Imports LidorSystems.IntegralUI.Lists

. . .

Private Sub textBox1_DragOver(ByVal sender As Object, ByVal e As DragEventArgs)
    If e.Data.GetDataPresent(GetType(ListBoxItem)) Then
        e.Effect = DragDropEffects.Move
    Else
        e.Effect = DragDropEffects.None
    End If
End Sub

```

```

Private Sub textBox1_DragDrop(ByVal sender As Object, ByVal e As DragEventArgs)
    If e.Data.GetDataPresent(GetType(ListBoxItem)) Then
        ' Get the dragged item
        Dim item As ListBoxItem = DirectCast(e.Data.GetData(GetType(ListBoxItem)),
        ListBoxItem)

        ' Set the Text property of TextBox control to contain the item text
        Me.textBox1.Text = item.Text
    End If
End Sub

[C#]
using LidorSystems.IntegralUI.Lists;

. . .

private void textBox1_DragOver(object sender, DragEventArgs e)
{
    if (e.Data.GetDataPresent(typeof(ListBoxItem)))
        e.Effect = DragDropEffects.Move;
    else
        e.Effect = DragDropEffects.None;
}

private void textBox1_DragDrop(object sender, DragEventArgs e)
{
    if (e.Data.GetDataPresent(typeof(ListBoxItem)))
    {
        // Get the dragged item
        ListBoxItem item = (ListBoxItem)e.Data.GetData(typeof(ListBoxItem));

        // Set the Text property of TextBox control to contain the item text
        this.textBox1.Text = item.Text;
    }
}

```

Sorting

When you have large list of items, the best to search through them is if they are sorted. The ListBox control supports predefined sorting of items based on some predefined types. Also, you can customize the sorting by creating your own implementation of IComparer interface.

Typically items are sorted using the **Sorting** property, which can have three values: None, Ascending and Descending. By default, the Sorting property has value None, which means that automatically sorting of items is disabled.

Use of predefined sorting method

When the Sorting property is set to other value than None, automatically sorting is active. So whenever a new item is added or removed, the list will be sorted. The items will be sorted depending of current value of ComparerObjectType, which can have one of these values:

- Double
- Integer
- String

Additionally, the sorting will only work for those items that have their **SortTag** set to some value. Items with their SortTag set to null, will be excluded from sorting.

In the following example the items will be sorted by their Integer value from the largest to lowest number:

```
[VB]
' At first set the SortTag for each node to some Integer value
For i As Integer = 0 To Me.listBox1.Items.Count - 1
    Me.listBox1.Items(i).SortTag = i
Next

' Change the ComparerObjectType to Integer, so the sorting will be used
comparing Integer values
Me.listBox1.ComparerObjectType =
LidorSystems.IntegralUI.Lists.ComparerObjectType.Integer

' Arrange items from largest to lowest number
Me.listBox1.Sorting = SortOrder.Descending

[C#]
// At first set the SortTag for each item to some Integer value
for (int i = 0; i < this.listBox1.Items.Count; i++)
    this.listBox1.Items[i].SortTag = i;

// Change the ComparerObjectType to Integer, so the sorting will be used
comparing Integer values
this.listBox1.ComparerObjectType =
LidorSystems.IntegralUI.Lists.ComparerObjectType.Integer;

// Arrange items from largest to lowest number
this.listBox1.Sorting = SortOrder.Descending;
```

Create custom sorting

To customize the sort order, you must write a class that implements the **IComparer** interface and set the **ListItemSorter** property to an object of that class. This is useful, for example, when you want to sort items by DateTime value added to the item Tag property.

Here is an example of custom sort operation. At first create a list of items:

```
[VB]
Imports LidorSystems.IntegralUI.Lists

. . .

Dim item As ListBoxItem = Nothing
For i As Integer = 0 To 9
    item = New ListBoxItem("Item " & i.ToString())
    item.Tag = DateTime.Now

    Me.listBox1.Items.Add(item)
Next

Me.listBox1.UpdateLayout()
```

```
[C#]
using LidorSystems.IntegralUI.Lists;

. . .

ListBoxItem item = null;
for (int i = 0; i < 10; i++)
{
    item = new ListBoxItem("Item " + i.ToString());
    item.Tag = DateTime.Now;

    this.listBox1.Items.Add(item);
}

this.listBox1.UpdateLayout();
```

Then create a new class that implements the IComparer interface:

```
[VB]
Public Class CustomItemComparer
    Implements System.Collections.IComparer
    Private sortType As System.Windows.Forms.SortOrder =
System.Windows.Forms.SortOrder.Ascending

    Public Sub New()
    End Sub

    Public Sub New(ByVal order As System.Windows.Forms.SortOrder)
        sortType = order
    End Sub

    Public Function Compare(ByVal x As Object, ByVal y As Object) As Integer
        If sortType = System.Windows.Forms.SortOrder.Descending Then
            Dim temp As Object = x
            x = y
            y = temp
        End If

        Dim firstDate As Object = TryCast(x, ListBoxItem).Tag
        Dim secondDate As Object = TryCast(y, ListBoxItem).Tag

        If firstDate IsNot Nothing AndAlso secondDate IsNot Nothing Then
            If firstDate.[GetType]() Is GetType(DateTime) AndAlso secondDate.
[GetType]() Is GetType(DateTime) Then
                Return DateTime.Compare(DirectCast(firstDate, DateTime),
DirectCast(secondDate, DateTime))
            End If
        End If

        Return 0
    End Function
End Class

[C#]
public class CustomItemComparer : System.Collections.IComparer
{
    private System.Windows.Forms.SortOrder sortType =
System.Windows.Forms.SortOrder.Ascending;
```

```

public CustomItemComparer()
{
}

public CustomItemComparer(System.Windows.Forms.SortOrder order)
{
    sortType = order;
}

public int Compare(object x, object y)
{
    if (sortType == System.Windows.Forms.SortOrder.Descending)
    {
        object temp = x;
        x = y;
        y = temp;
    }

    object firstDate = (x as ListBoxItem).Tag;
    object secondDate = (y as ListBoxItem).Tag;

    if (firstDate != null && secondDate != null)
    {
        if (firstDate.GetType() == typeof(DateTime) && secondDate.GetType() ==
typeof(DateTime))
            return DateTime.Compare((DateTime)firstDate, (DateTime)secondDate);
    }

    return 0;
}
}

```

At the end, add an object from this class to the ListItemSorter property:

```

[VB]
Me.listBox1.ListItemSorter = New CustomItemComparer(SortOrder.Descending)

[C#]
this.listBox1.ListItemSorter = new CustomItemComparer(SortOrder.Descending);

```

Search

How to find an item by using specific criteria

If you want to find an item with specific value, the best is to use the FindItem method. The search criteria can be either byKey or byTagString. Here is an example:

```

[VB]
Dim item As ListBoxItem = Me.listBox1.FindItem("ITM", ListSearchCriteria.byKey)

[C#]
ListBoxItem item = this.listBox1.FindItem("ITM", ListSearchCriteria.byKey);

```

The method will search through items, and compare their Key property value to match the "ITM" value. If some item is found it will return their reference, otherwise a null will be returned.

How to get an item reference from mouse position

Sometimes, while hovering inside the ListBox control client area, you may want to know which item the actual mouse cursor points at. To receive the item reference, you need to use the `GetItemAt` method, here is how:

```
[VB]
' Convert the screen coordinates of the mouse cursor position to client
coordinates of the ListBox control
Dim mousePos As Point =
Me.listBox1.ContentPanel.PointToClient(Control.MousePosition)

' Get the item reference (if there is any), at the specified position
Dim item As ListBoxItem = Me.listBox1.GetItemAt(mousePos)

[C#]
// Convert the screen coordinates of the mouse cursor position to client
coordinates of the ListBox control
Point mousePos =
this.listBox1.ContentPanel.PointToClient(Control.MousePosition);

// Get the item reference (if there is any), at the specified position
ListBoxItem item = this.listBox1.GetItemAt(mousePos);
```

Keyword search

There is a built-in search operations by pressing keys. For this behavior, the **KeySearchMode** property is used with following values:

- None – The search is disabled.
- Partial – The search is started whenever there is a key(s) pressed.
- Full – The same as Partial search. In other list controls, this mode has different purpose.

The search process begins whenever there is a single key pressed or a string with multiple characters in it. The search begins always from index 0. For example: whenever the 'A' key is pressed, the search is started. Also if you pressed keys in following order 'ABC' the search is also started. However, you must press the consecutive keys in time less than **500 milliseconds** between **each** pressed key. This is enough time to enter a search string with large length. If the item is found it is selected and positioned to the center of control visible area.

You can also manually search for items, by using `FindItem` methods. Here is how:

1. Set the **KeySearchMode** to **None**
2. Handle the **KeyDown** event or use some other way to start the search (like Button click event)
3. Use the **FindItem** methods

```
[VB]
Private Sub listBox1_KeyDown(ByVal sender As Object, ByVal e As KeyEventArgs)
' Search for an item with matching text starting from index 0
Dim item As LidorSystems.IntegralUI.Lists.ListBoxItem =
Me.listBox1.FindItem(e.KeyCode.ToString(), 0)

If item IsNot Nothing Then
' Select the found item
```

```

        Me.listBox1.SelectedItem = item

        If Me.listBox1.IsVScrollVisible() Then
            ' If the item is not in visible area, set the scrollbar position to
show
            ' the found item in center of ListView control
            If item.Bounds.Y > Me.listBox1.ContentPanel.ClientRectangle.Height / 2
Then
                Me.listBox1.SetScrollPos(New Point(0, item.Bounds.Y -
Me.listBox1.ContentPanel.ClientRectangle.Height / 2))
            Else
                Me.listBox1.SetScrollPos(New Point(0, item.Bounds.Y -
item.Bounds.Height))
            End If
        End If
    End If
End Sub

[C#]
private void listBox1_KeyDown(object sender, KeyEventArgs e)
{
    // Search for an item with matching text starting from index 0
    LidorSystems.IntegralUI.Lists.ListBoxItem item =
this.listBox1.FindItem(e.KeyCode.ToString(), 0, true);

    if (item != null)
    {
        // Select the found item
        this.listBox1.SelectedItem = item;

        if (this.listBox1.IsVScrollVisible())
        {
            // If the item is not in visible area, set the scrollbar position to
show
            // the found item in center of ListView control
            if (item.Bounds.Y > this.listBox1.ContentPanel.ClientRectangle.Height /
2)
                this.listBox1.SetScrollPos(new Point(0, item.Bounds.Y -
this.listBox1.ContentPanel.ClientRectangle.Height / 2));
            else
                this.listBox1.SetScrollPos(new Point(0, item.Bounds.Y -
item.Bounds.Height));
        }
    }
}

```

How to maintain scroll position

In order to manually change the position of scrollbar, there are two methods for this purpose:

- GetScrollPos – returns the current position of vertical and horizontal scrollbar
- SetScrollPos – sets the position of vertical and horizontal scrollbar to a specified value

Every item has Bounds property which holds the area in which the item content is drawn. This property has their position in absolute coordinates related to the origin of the control client area. So, if you want to have some specific item placed at the top of visible area, you can do that by using this code:

```
[VB]  
Me.listBox1.SetScrollPos (New Point (0, item.Bounds.Y))
```

```
[C#]  
this.listBox1.SetScrollPos (new Point (0, item.Bounds.Y));
```

The Top position of the item is directly related to the position of vertical scrollbar.

XML Encoding

By default the ListBox shows items ordered in simplified way, a vertical list. This is not much useful when you want to present your data in more custom layout.

By using XML tags you can create various different templates which will contain custom objects placed in custom location in item space. These custom objects represents: text, images, custom controls, hyperlinks, animated gifs etc. There are a set of XML tags with which you can design the layouts for every item. These templates can be applied to all items (to keep uniform look for every item), or separate template can be applied for every specific item.

The result is a very rich and customizable presentation of your data.

XML Tags that are supported

The use of XML tags is very like the using of standard HTML tags. In the following section the list of supported XML tags is presented in alphabetical order:

Tag	AtrIBUTE	Description
<a>	href id style underlined	Defines an anchor with which you can create a link to another document The target URL of the link The identifier of the hyperlink An inline style definition Determines whether the link text is underlined. Default value is true.
	id style	The text is rendered as bold The identifier of the text enclosed with this tag An inline style definition
 		inserts a single line break
<control>	height index style width	Defines a custom control Sets the height of a control Specifies the position of the control in a collection An inline style definition Sets the width of a control
<div>	style	Defines the main section of the content An inline style definition
	color face id size style	specifies the font of text Defines the color of the text Defines the font name of the text The identifier of the text enclosed with this tag Defines the size of the text An inline style definition
<i>		The text is rendered as italic

	id style	The identifier of the text enclosed with this tag An inline style definition
	assemblypath height id index resource src style width	Defines an image Specifies the path of an assembly used as a resource file Sets the height of an image The identifier of the image Specifies the position of the image in a collection The resource object to be retrieved from assembly The URL of the image to display An inline style definition Sets the width of an image
<p>	indent style	Defines a new section (paragraph) in the content Specifies a space (in pixels) by which paragraph's content is moved to the right An inline style definition
<table>	cellpadding cellspacing style width	defines a table Specifies the space between the table cell border and content Specifies the space between table cells An inline style definition Specifies the width of the table. Can be specified in % or pixels.
<td>	align colspan height rowspan style valign width	Defines a cell in a table Specifies the horizontal alignment of cell content Indicates the number of columns this cell should span Specifies the height of the table cell. Can be specified in % or pixels. Indicates the number of rows this cell should span An inline style definition Specifies the vertical alignment of cell content Specifies the width of the table cell. Can be specified in % or pixels.
<tr>	align height style valign	Defines a new section (paragraph) in the content Specifies the horizontal alignment of cell content Specifies the height of the table row. Can be specified in % or pixels. An inline style definition Specifies the vertical alignment of cell content
<r>	id style	Defines regular text The identifier of the text enclosed with this tag An inline style definition
<s>	id	Defines strikethrough text The identifier of the text enclosed with this tag

	style	An inline style definition
<style>		Defines a style used for rendering and formatting of the content
	align	Specifies the content alignment
	background	Specifies the color of the object background
	background-color	Specifies the fading color of the object background
	border-color	Specifies the color of the object border
	fill-style	Specifies the type of gradient fill of the object background
	font	Specifies the color of the text
	hover-color	Specifies the color of the object background while mouse cursor hovers over it
	background-color	Specifies the fading color of the object background while mouse cursor hovers over it
	hover-text-color	Specifies the color of the text while mouse cursor hovers over it
	id	The identifier of the style
	margin	Defines the space around an object's border
	padding	Defines the space between the border and the content of an object
	rendering	Specifies the rendering mode for text
selected-text-color	Specifies the color of the text when it is selected	
text-color	Specifies the color of the text	
transparency	Specifies the percentage of transparency	
<u>		defines underlined text
	id	The identifier of the text enclosed with this tag
	style	An inline style definition

In the following section we will present you how to use these tags in various combinations. As a result there would be created custom layouts for item's content.

Working with containers: <div> and <p> tags

The simplest form of formatted content is constructed by using the <div> tag and sample text. Here is an example:

```
[VB]
item.Content = "<div>Single text line</div>"

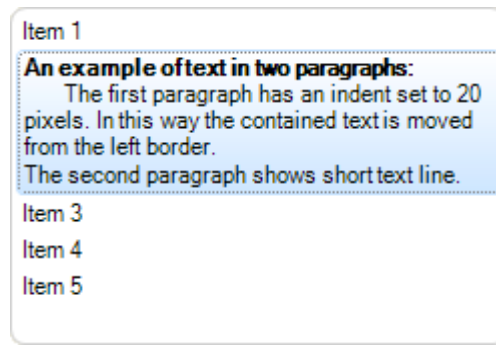
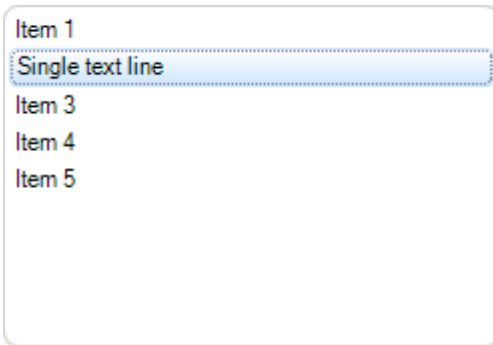
[C#]
item.Content = "<div>Single text line</div>";
```

In some cases you want to display text in multiple lines or in different paragraphs. For this situation the <p> tag is used. Here is an example:

```
[VB]
item.Content = "<div><b>An example of text in two paragraphs:</b><p
indent=""20"">The first paragraph has an indent set to 20 pixels. In this way
the contained text is moved from the left border.</p><p>The second paragraph
shows short text line.</p></div>"

[C#]
item.Content = "<div><b>An example of text in two paragraphs:</b><p
indent=\"20\">The first paragraph has an indent set to 20 pixels. In this way
the contained text is moved from the left border.</p><p>The second paragraph
shows short text line.</p></div>";
```

The result is shown in following pictures:

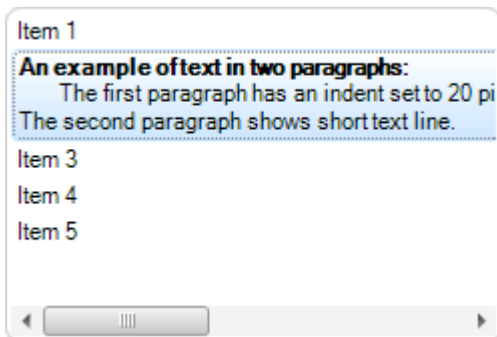


From the pictures you can also notice that the text is wrapped in multiple lines when words are reaching the end of the item content space. This happens because the WordWrap property by default is set to True for all items in the ListBox control.

If for example we set the WordWrap property for this item to False, the resulting look is different.

```
[VB]
item.WordWrap = False
item.Content = "<div><b>An example of text in two paragraphs:</b><p
indent=""20"">The first paragraph has an indent set to 20 pixels. In this way
the contained text is moved from the left border.</p><p>The second paragraph
shows short text line.</p></div>"

[C#]
item.WordWrap = false;
item.Content = "<div><b>An example of text in two paragraphs:</b><p
indent=\"20\">The first paragraph has an indent set to 20 pixels. In this way
the contained text is moved from the left border.</p><p>The second paragraph
shows short text line.</p></div>";
```



More information about word wrapping can be found in "Using word wrap" section of this document.

Working with tag

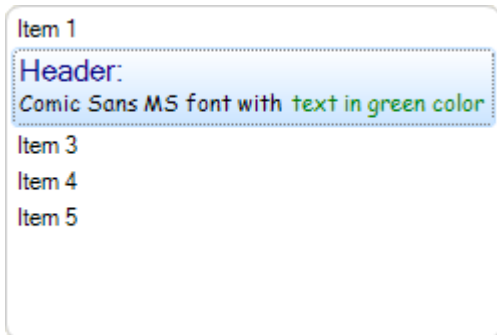
Normally some parts of the text shown in a single content can have different font or text size. In order to do that the tag is used.

In following example we will present you how to create content with header, text in multiple lines with some parts set to different font and color.

```
[VB]
item.Content = "<div><font size=""11"" color=""#000080"" >Header:</font><p><font
face=""Comic Sans MS"">Comic Sans MS font with<font color=""Green""> text in
green color</font></font></p></div>"

[C#]
item.Content = "<div><font size=\"11\" color=\"#000080\" >Header:</font><p><font
face=\"Comic Sans MS\">Comic Sans MS font with<font color=\"Green\"> text in
green color</font></font></p></div>";
```

The result is a text in two lines. First line is shown in **Microsoft Sans Serif 11pt** in blue color, and the second line is shown in **Comic Sans MS** font containing parts in green color:

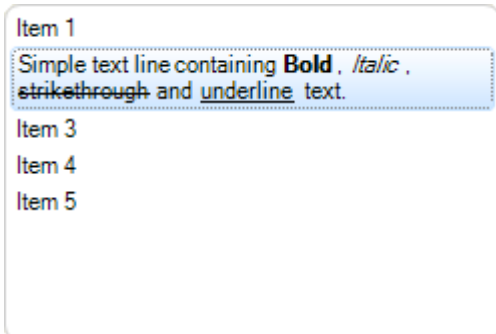


Working with font style tags

In font style tags group are , <i>, <s> and <u>. With these tags you can change the style of font for some part or whole text. Here is an example:

```
[VB]
item.Content = "<div>Simple text line containing <b>Bold</b>, <i>Italic</i>,
<s>strikethrough </s>and <u>underline</u> text.</div>"

[C#]
item.Content = "<div>Simple text line containing <b>Bold</b>, <i>Italic</i>,
<s>strikethrough </s>and <u>underline</u> text.</div>";
```



Working with <a> tag

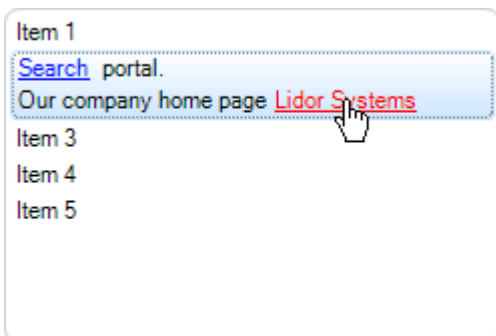
One of objects that you can create in content is a hyperlink object. The link you are entering can be some file or page located on the web or on local hard disk.

If the link points to some application file like Notepad.exe, you can open it by simple clicking on the hyperlink. However in order this to work, you will also need to handle ItemObjectClicked event (see the "How to handle events" section in the beginning of this document).

Here is an example of content with two hyperlinks:

```
[VB]
item.Content = "<div><a href=""www.google.com""
style=""textcolor:Blue"">Search</a> portal. <br></br>Our company home page <a
href=""www.lidorsystems.com"" style=""textcolor:Red"">Lidor Systems</a></div>";

[C#]
item.Content = "<div><a href=""www.google.com""
style=""textcolor:Blue"">Search</a> portal. <br></br>Our company home page <a
href=""www.lidorsystems.com"" style=""textcolor:Red"">Lidor Systems</a></div>";
```



As you can see from the picture, the hyperlinks have also different colors. Also, the hand cursor is automatically shown when the mouse hovers over the hyperlink. In future versions we may extend this to be able to add custom cursors.

In this example we have used the style attribute in which you can set different sub attributes for the object to which this style is applied. See below for more information how to use styles.

Working with tag

With this tag you can add images located on some local location or there is an option to create ImageList and extract the image from the list by using specified index.

An example how to create content with two images located on local hard disk:

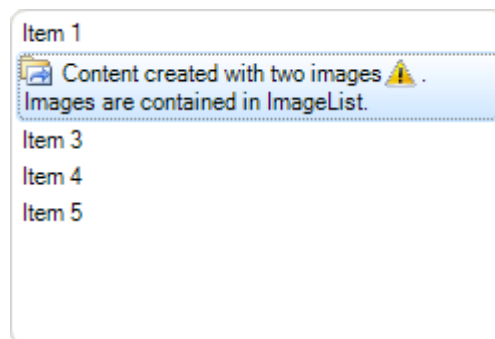
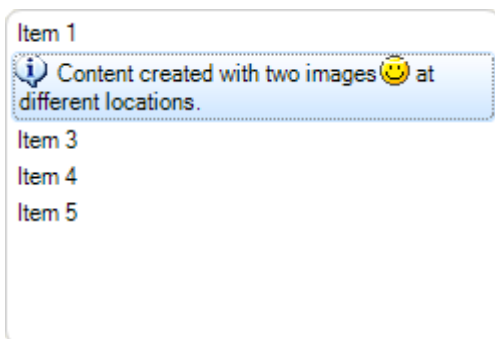
```
[VB]
item.Content = "<div><img src=\"C:\\images\\info.ico\"></img> Content created
with two images <img src=\"C:\\images\\smiley.ico\"></img>at different
locations.</div>"

[C#]
item.Content = "<div><img src=\"C:\\images\\info.ico\"></img> Content created
with two images <img src=\"C:\\images\\smiley.ico\"></img>at different
locations.</div>";
```

An example how to create content with two images located in ImageList referenced by ListBox control:

```
[VB]
item.Content = "<div><img index=\"0\"></img> Content created with two images
<img index=\"1\"></img>. Images are contained in ImageList.</div>";

[C#]
item.Content = "<div><img index=\"0\"></img> Content created with two images
<img index=\"1\"></img>. Images are contained in ImageList.</div>";
```



By default the images have their original size. You can change their size by setting the width and height attributes of the tag. Also you can apply margin and padding attributes by setting the style attribute values (see below how to use styles).

Working with <control> tag

With text, images and hyperlinks you can create very rich content. However, placing custom controls will give more interaction to the user.

The <control> tag is specifically added to meet this task. Every custom control, standard or third party, can be placed in single content of the item. You can add even complex controls like Grid.

Here is an example on how to add a button and a checkbox to item content:

```
[VB]
' At first the control must be created and added to
' the item Controls collection
```

```

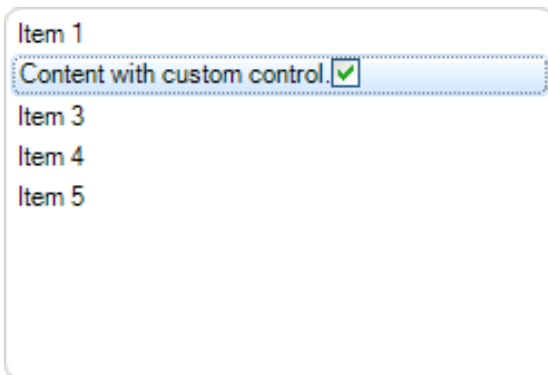
Dim cBox As New CheckBox()
cBox.Size = New Size(14, 14)
Me.listBox1.SelectedItem.Controls.Add(cBox)

' In <control> tag use the index to reference the control
Me.listBox1.SelectedItem.Content = "<div>Content with custom control. <control
index=""0""></control></div>"

[C#]
// At first the control must be created and added to
// the item Controls collection
CheckBox cBox = new CheckBox();
cBox.Size = new Size(14, 14);
this.listBox1.SelectedItem.Controls.Add(cBox);

// In <control> tag use the index to reference the control
this.listBox1.SelectedItem.Content = "<div>Content with custom control. <control
index=""0""></control></div>";

```



The use of controls in simple format is not very useful. The controls like text or any other object, can also be wrapped and shown in next line, but that is all. Mainly in simple form there are no attributes by which you can place the control at specific place or aligned it to the right side of the content space, for example.

This can be done by using table formatting which is explained in next section.

Working with <table>, <tr> and <td> tags

The best way to create custom layouts with different objects placed at different locations in the content is by using table formatting.

One of the reasons the table is used is presentation of data in ordered way for all items. In the following example we will show you how to place a text, hyperlink and custom control in a content using <table> tag:

```

[VB]
Dim cBox As New CheckBox()
cBox.Size = New Size(14, 14)
Me.listBox1.SelectedItem.Controls.Add(cBox)

item.Content = "<div><table cellpadding=""1"" cellspacing=""2""><tr><td>Simple
text line</td><td><a href=""www.lidorsystems.com"">More
info</a></td><td><control index=""0""></control></td></tr></table></div>"

```

```
[C#]
CheckBox cBox = new CheckBox();
cBox.Size = new Size(14, 14);
this.listBox1.SelectedItem.Controls.Add(cBox);

item.Content = "<div><table cellspacing=\"2\" cellpadding=\"1\"><tr><td>Simple
text line</td><td><a href=\"www.lidorsystems.com\">More
info</a></td><td><control index=\"0\"></control></td></tr></table></div>";
```

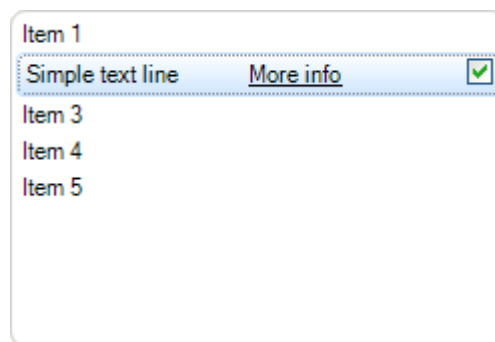
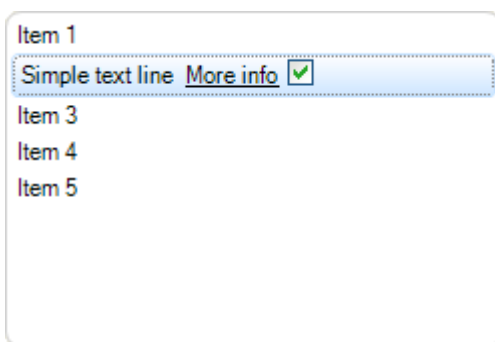
In this example all three objects are placed next to each other, which doesn't give the best look. Also, it acts like there is not table present. We can change that by specifying the width for each table cell. If the width is not specified the table cell will have width as the minimum width of their content.

```
[VB]
Dim cBox As New CheckBox()
cBox.Size = New Size(14, 14)
Me.listBox1.SelectedItem.Controls.Add(cBox)

item.Content = "<div><table cellspacing=\"2\" cellpadding=\"1\"
width=\"100%\"><tr><td width=\"50%\">Simple text line</td><td width=\"50%\"><a
href=\"www.lidorsystems.com\">More info</a></td><td><control
index=\"0\"></control></td></tr></table></div>"

[C#]
CheckBox cBox = new CheckBox();
cBox.Size = new Size(14, 14);
this.listBox1.SelectedItem.Controls.Add(cBox);

item.Content = "<div><table cellspacing=\"2\" cellpadding=\"1\"
width=\"100%\"><tr><td width=\"50%\">Simple text line</td><td width=\"50%\"><a
href=\"www.lidorsystems.com\">More info</a></td><td><control index=\"0\">
</control></td></tr></table></div>";
```



The change is obvious. By specifying the width of table cell in percents we have achieved the cell to shrink or expand depending of the current width of the item content. For the last cell the width is not specified. This is done to make sure that the CheckBox will remain always docked at the right side of the content.

Here is a more complex layout, which includes column and row span:

```

[VB]
' Use a temporary variable for storing large string
Dim content As String = "<div><table width=""100%"">"
' Add the first row with four cells
content += "<tr><td rowspan=""2""><img index=""1""></img></td><td
width=""50%"">First line</td><td width=""50%""><a href=""www.lidorsystems.com""
style=""textcolor:Blue"">More info</a></td><td><control index=""0""></control></
td></tr>"
' Add the second row and span the cell over three other cells
content += "<tr><td colspan=""3""><i>Second text line which is a little
longer.</i></td></tr>"
' Close the table and div tags
content += "</table></div>"

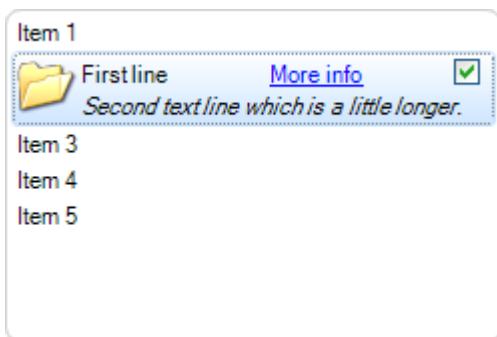
' Add the content string to the item Content property
item.Content = content

[C#]
// Use a temporary variable for storing large string
String content = "<div><table width=""100%"">";
// Add the first row with four cells
content += "<tr><td rowspan=""2""><img index=""1""></img></td><td
width=""50%"">First line</td><td width=""50%""><a href=""www.lidorsystems.com""
style=""textcolor:Blue"">More info</a></td><td><control index=""0""></control></
td></tr>";
// Add the second row and span the cell over three other cells
content += "<tr><td colspan=""3""><i>Second text line which is a little
longer.</i></td></tr>";
// Close the table and div tags
content += "</table></div>";

// Add the content string to the item Content property
item.Content = content;

```

The first cell is spanned across two rows. The second cell of the second row is spanned across three columns. The result is shown in following picture:



Working with <style> tag

The styles are very powerful tool if you use them correctly. You can create several styles in beginning and later used them for specific parts of the code only by calling their Id. Also you can use inline coding to apply specific style to some part of the content.

Here is how to create styles at beginning of the code:

Here is how to create styles at beginning of the code:

[VB]

```
Dim content As String = "<div><style id=""1"" textcolor=""Red""  
font=""name:Arial;size:10;style:b""></style>"  
content += "<style id=""2"" textcolor=""Blue""  
font=""name:Verdana;size:8;style:u""></style>"  
content += "<style id=""3"" textcolor=""Green"" font=""name:Times New  
Roman;size:8;style:i""></style>"  
content += "<p style=""id:1"">The first text line in red.</p>"  
content += "<p style=""id:2"">The second in blue, <font style=""id:3"">with  
parts in green.</font></p>"  
content += "</div>"
```

```
item.Content = content
```

[C#]

```
String content = "<div><style id=""1"" textcolor=""Red""  
font=""name:Arial;size:10;style:b""></style>;"  
content += "<style id=""2"" textcolor=""Blue""  
font=""name:Verdana;size:8;style:u""></style>;"  
content += "<style id=""3"" textcolor=""Green"" font=""name:Times New  
Roman;size:8;style:i""></style>;"  
content += "<p style=""id:1"">The first text line in red.</p>;"  
content += "<p style=""id:2"">The second in blue, <font style=""id:3"">with  
parts in green.</font></p>;"  
content += "</div>;";
```

```
item.Content = content;
```

Item 1

The first text line in red.

The second in blue, with parts in green.

Item 3

Item 4

Item 5

Here is how to use inline style definition:

```

[VB]
Dim content As String = "<div>";
content += "<p style=""textcolor:Red;font:Arial,10,b"">The first text line in
red.</p>";
content += "<p style=""textcolor:Blue;font:Verdana,8,u"">The second in blue,
<font style=""textcolor:Green;font:Times New Roman,8,i"">with parts in
green.</font></p>";
content += "</div>";

item.Content = content

[C#]
String content = "<div>";
content += "<p style=\"textcolor:Red;font:Arial,10,b\">The first text line in
red.</p>";
content += "<p style=\"textcolor:Blue;font:Verdana,8,u\">The second in blue,
<font style=\"textcolor:Green;font:Times New Roman,8,i\">with parts in
green.</font></p>";
content += "</div>";

item.Content = content;

```

The result is the same as in previous picture, only the use of styles has changed. Instead of using predefined styles, we have used inline style definitions to change different parts of the text.

Styles can be applied for every object, not just text. For example the Margin and Padding attributes of the style, has more value when paragraphs or tables are used. The same applies for content alignment which has more value when it is used with tables.

Using word wrap

By default the WordWrap for the ListBox control and all items is set to True. This means that whenever some object reach the end of the line, if there is not enough space for the object to be placed at the line end, it will be placed in beginning of a new line. In this way by changing the width of the ListBox control the content of their items will expand or collapse , but always will show their full content in visible area of the control.

If you don't want to use word wrapping, then the item content will remain in exact place as item layout is constructed. Meaning, some objects may be placed outside the visible area of the ListBox control. This will trigger appearance of horizontal or vertical scrollbar, which allows you to scroll the visible area in order to show these objects.

As stated above, all items can have their independent setting of word wrap. This is very useful to have, when designing different layouts for different items. Some of them can have word wrapping while others don't. You only need to change the WordWrap property for specific items.

Serialization

You can use serialization to memorize the current state of ListBox. Also, this feature is very useful in creation of color schemes.

How to serialize the control content in Streams

In order to serialize the control content in streams, there are two methods:

- LoadFromStream – load the control content from Stream
- SaveToStream – save the control content in Stream

Here is how you can use these methods:

```
[VB]
Imports LidorSystems.IntegralUI.Serialization

. . .

' Create a MemoryStream object
Private memStream As New MemoryStream()

' Click on the btnLoad button to load the content of MemoryStream object
' to ListBox control
Private Sub btnLoad_Click(ByVal sender As Object, ByVal e As System.EventArgs)
    Dim serializer As New ListBoxSerializer()
    memStream.Seek(0, SeekOrigin.Begin)
    serializer.LoadFromStream(Me.listBox1, memStream)
End Sub

' Click on the btnSave button to save the control content to
' MemoryStream object
Private Sub btnSave_Click(ByVal sender As Object, ByVal e As System.EventArgs)
    Dim serializer As New ListBoxSerializer()
    serializer.SaveToStream(Me.listBox1, memStream, False)
End Sub

[C#]

using LidorSystems.IntegralUI.Serialization;

. . .

// Create a MemoryStream object
MemoryStream memStream = new MemoryStream();

// Click on the btnLoad button to load the content of MemoryStream object
// to ListBox control
private void btnLoad_Click(object sender, System.EventArgs e)
{
    ListBoxSerializer serializer = new ListBoxSerializer();
    memStream.Seek(0, SeekOrigin.Begin);
    serializer.LoadFromStream(this.listBox1, memStream);
}

// Click on the btnSave button to save the control content to
// MemoryStream object
```

```
private void btnSave_Click(object sender, System.EventArgs e)
{
    ListBoxSerializer serializer = new ListBoxSerializer();
    serializer.SaveToStream(this.listBox1, memStream, false);
}
```

How to serialize the control content in files

In order to serialize the control content in files, there are two ways:

- By using file streams
- By using file names

Serialization using file streams

```
[VB]
Imports LidorSystems.IntegralUI.Serialization

. . .

' Click on the btnLoad button to load the content of FileStream object
' to ListBox control
Private Sub btnLoad_Click(ByVal sender As Object, ByVal e As System.EventArgs)
    Dim serializer As New ListBoxSerializer()
    Dim fs As FileStream = File.OpenRead("C:\fs.xml")
    serializer.LoadFromStream(Me.listBox1, fs)
End Sub

' Click on the btnSave button to save the control content to
' FileStream object
Private Sub btnSave_Click(ByVal sender As Object, ByVal e As System.EventArgs)
    Dim serializer As New ListBoxSerializer()
    Dim fs As FileStream = File.Create("C:\fs.xml")
    serializer.SaveToStream(Me.listBox1, fs, True)
End Sub

[C#]

using LidorSystems.IntegralUI.Serialization;

. . .

// Click on the btnLoad button to load the content of FileStream object
// to ListBox control
private void btnLoad_Click(object sender, System.EventArgs e)
{
    ListBoxSerializer serializer = new ListBoxSerializer();
    FileStream fs = File.OpenRead("C:\\fs.xml");
    serializer.LoadFromStream(this.listBox1, fs);
}

// Click on the btnSave button to save the control content to
// FileStream object
private void btnSave_Click(object sender, System.EventArgs e)
{
    ListBoxSerializer serializer = new ListBoxSerializer();
    FileStream fs = File.Create("C:\\fs.xml");
```

```

        serializer.SaveToStream(this.listBox1, fs, true);
    }

```

Serialization using file names

In order to serialize the control content in files, there are two methods:

- LoadFromFile – load the control content from file
- SaveToFile – save the control content in file

```

[VB]
' Click on the btnLoad button to open the Load dialog, by which you can
' choose the destination of the file
Private Sub btnLoad_Click(ByVal sender As Object, ByVal e As System.EventArgs)
    Dim openDlg As New OpenFileDialog()
    openDlg.Filter = "XML files (*.xml)|*.xml"
    openDlg.InitialDirectory = Application.ExecutablePath
    If openDlg.ShowDialog() = DialogResult.OK Then
        Me.listBox1.SuspendUpdate()

        Dim serializer As New
LidorSystems.IntegralUI.Serialization.ListBoxSerializer()
        serializer.LoadFromXml(Me.listBox1, openDlg.FileName)

        Me.listBox1.ResumeUpdate()
    End If
End Sub

' Click on the btnSave button to open the Save dialog, by which you can
' choose the destination of the file
Private Sub btnSave_Click(ByVal sender As Object, ByVal e As System.EventArgs)
    Dim saveDlg As New SaveFileDialog()
    saveDlg.Filter = "XML files (*.xml)|*.xml"
    saveDlg.AddExtension = True
    saveDlg.DefaultExt = ".xml"
    saveDlg.InitialDirectory = Application.ExecutablePath

    If saveDlg.ShowDialog() = DialogResult.OK Then
        Dim serializer As New
LidorSystems.IntegralUI.Serialization.ListBoxSerializer()
        serializer.SaveToXml(Me.listBox1, saveDlg.FileName)
    End If
End Sub

[C#]
using LidorSystems.IntegralUI.Serialization;

. . .

// Click on the btnLoad button to open the Load dialog, by which you can
// choose the destination of the file
private void btnLoad_Click(object sender, System.EventArgs e)
{
    OpenFileDialog openDlg = new OpenFileDialog();
    openDlg.Filter = "XML files (*.xml)|*.xml";
    openDlg.InitialDirectory = Application.ExecutablePath;
    if (openDlg.ShowDialog() == DialogResult.OK)

```

```

    {
        this.listBox1.SuspendUpdate();

        LidorSystems.IntegralUI.Serialization.ListBoxSerializer serializer = new
LidorSystems.IntegralUI.Serialization.ListBoxSerializer();
        serializer.LoadFromXml(this.listBox1, openFileDialog.FileName);

        this.listBox1.ResumeUpdate();
    }
}

// Click on the btnSave button to open the Save dialog, by which you can
// choose the destination of the file
private void btnSave_Click(object sender, System.EventArgs e)
{
    SaveFileDialog saveDlg = new SaveFileDialog();
    saveDlg.Filter = "XML files (*.xml)|*.xml";
    saveDlg.AddExtension = true;
    saveDlg.DefaultExt = ".xml";
    saveDlg.InitialDirectory = Application.ExecutablePath;

    if (saveDlg.ShowDialog() == DialogResult.OK)
    {
        LidorSystems.IntegralUI.Serialization.ListBoxSerializer serializer =
LidorSystems.IntegralUI.Serialization.new ListBoxSerializer();
        serializer.SaveToXml(this.listBox1, saveDlg.FileName);
    }
}

```

How to serialize the control content in SQL database

When you want to save the control content in some SQL database, at first you need a XML field in your table. The process of adding the control content to this field is the same as for streams, explained previously.

Loading content from a database and filling the ListBox control with it is also the same as using streams. But here, at first you need to create a MemoryObject:

```

[VB]
' Create a MemoryStream object
Dim memStream As New MemoryStream()

' sampleString is temporary variable that holds the database XML field content
' In this demonstration we are using some XML encoded text.
Dim sampleString As String = "<?xml version=""1.0"" encoding=""us-ascii""?
><ListBox Version=""1.0""><Items><Item Text=""Item 1""></Item></Items></ListBox>"

' Write the sampleString to the MemoryStream object
Dim sw As New StreamWriter(memStream, System.Text.Encoding.ASCII)
sw.Write(sampleString)
sw.Flush()

[C#]
// Create a MemoryStream object
MemoryStream memStream = new MemoryStream();

```

```
// sampleString is temporary variable that holds the database XML field content
// In this demonstration we are using some XML encoded text.
string sampleString = "<?xml version=\"1.0\" encoding=\"us-ascii\"?><ListBox
Version=\"1.0\"><Items><Item Text=\"Item 1\"></Item></Items></ListBox>";

// Write the sampleString to the MemoryStream object
StreamWriter sw = new StreamWriter(memStream, System.Text.Encoding.ASCII);
sw.Write(sampleString);
sw.Flush();
```

After that, you need to create a **ListBoxSerializer** object:

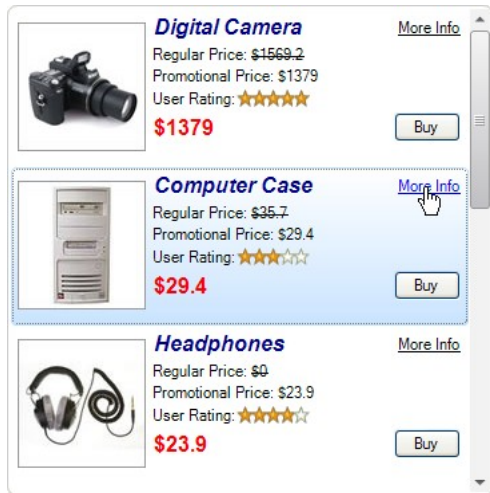
```
[VB]
Dim serializer As New ListBoxSerializer()
memStream.Seek(0, SeekOrigin.Begin)
serializer.LoadFromStream(Me.listBox1, memStream)

[C#]
ListBoxSerializer serializer = new ListBoxSerializer();
memStream.Seek(0, SeekOrigin.Begin);
serializer.LoadFromStream(this.listBox1, memStream);
```

As a result the control will be populated from the string placed in the MemoryStream object.

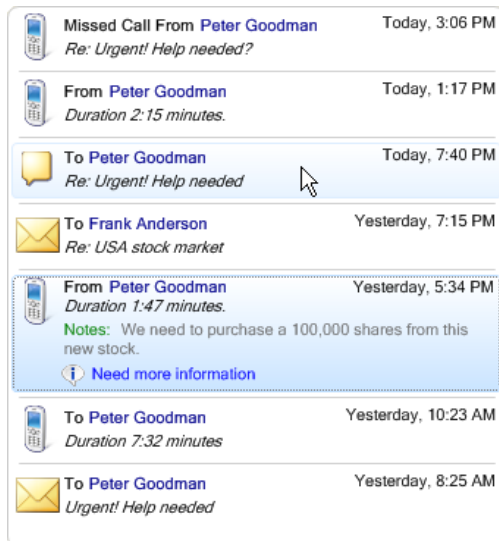
Sample projects

More information on creation of custom layouts by using XML tags can be found by examining the source code of the sample projects available on our web site:



Custom Catalog

Demonstrates how **custom objects** like text, images, hyperlinks and custom controls can be included in **IntegralUI ListBox**. Data is presented in two views, Simple and Extended, which shows how complex data formatting template can be applied for every item using various XML tags like tables, paragraphs, styles etc. The sample demonstrates also using wordwrap, three-state checkboxes and custom sort operations.



Messenger

Demonstrates how custom messenger application can be constructed by using special features of **IntegralUI ListBox** control. Various XML tags are used to build a different content for every item. Text are shown in different fonts and sizes, images are placed on custom locations, content alignment and table cell merging is also demonstrated. Items have different content in collapsed and expanded state.