Ribbon for WinForms

2021.12.21 更新

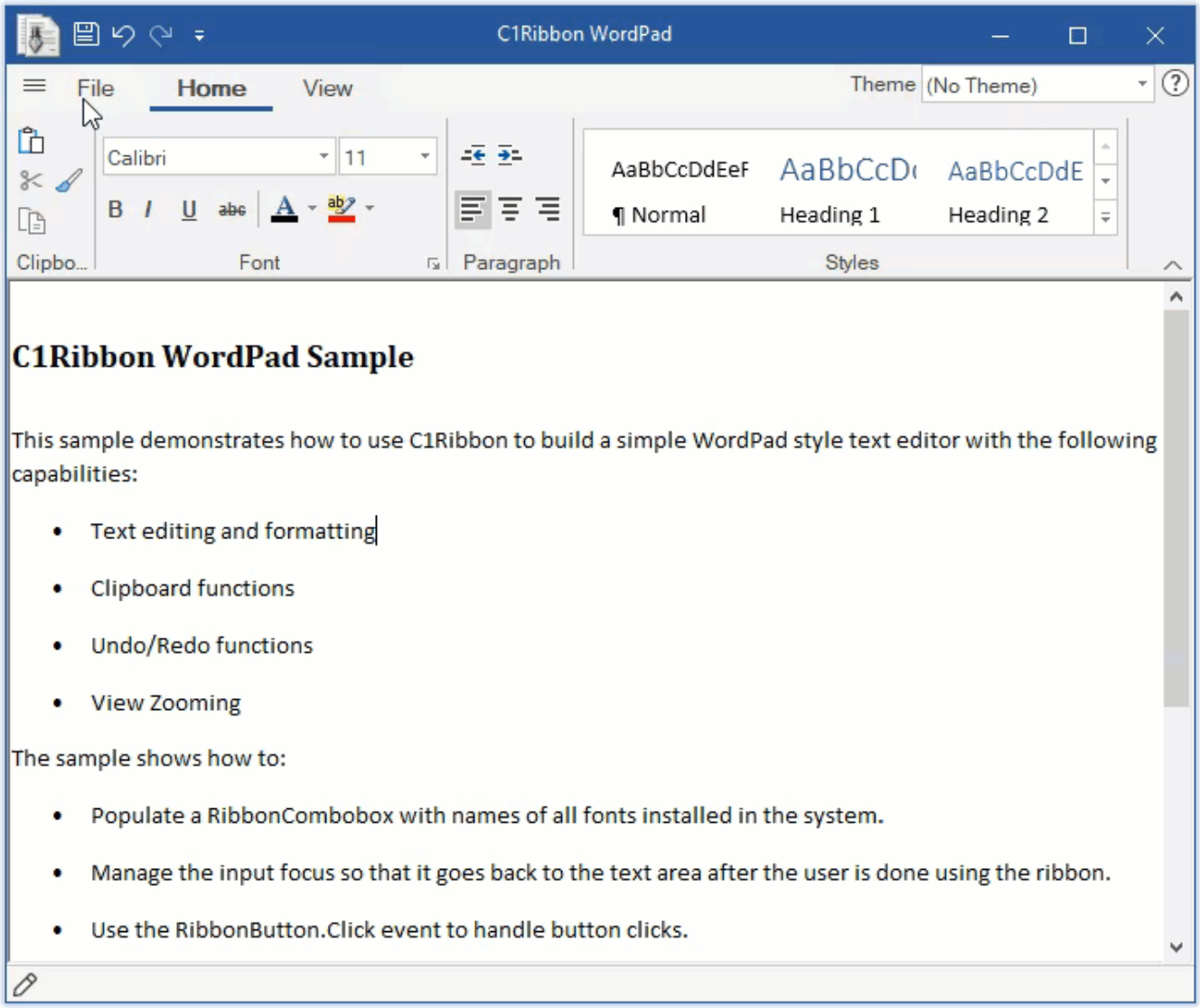
グレープシティ株式会社

目次

Ribbon の概要

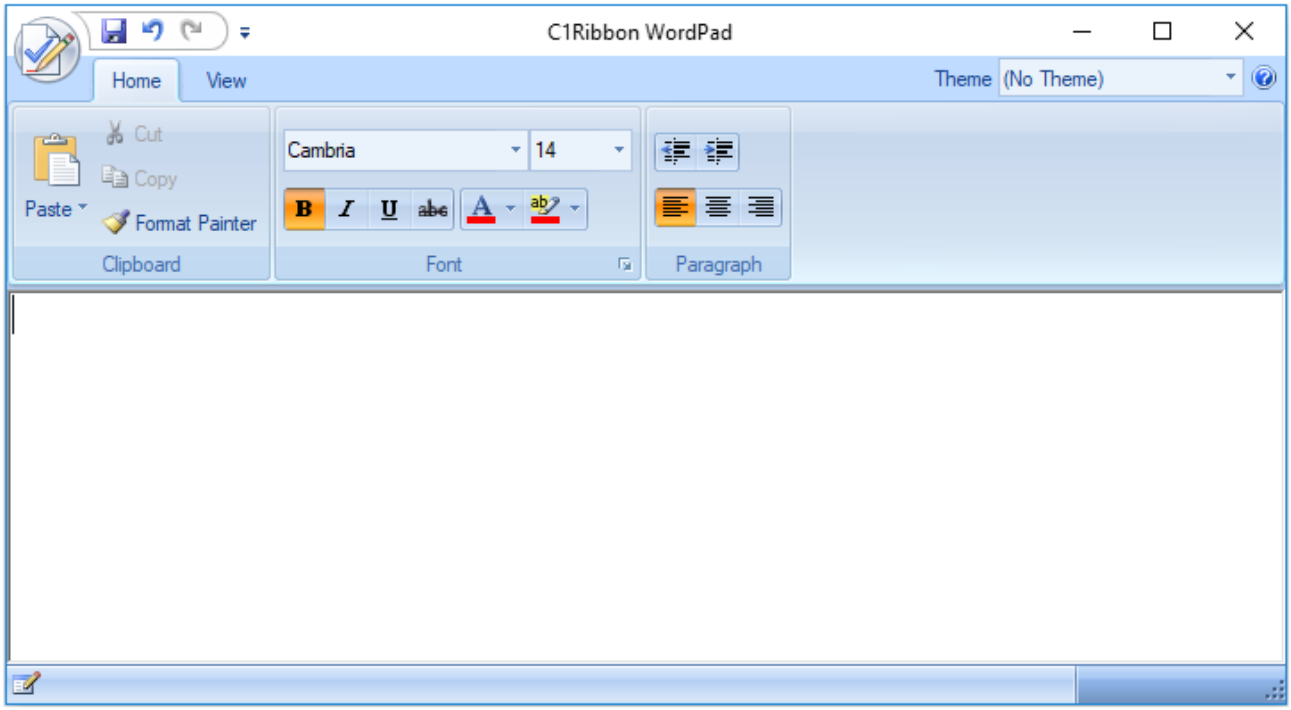
Ribbon for WinForms is a ribbon style flat menu similar to what Microsoft offers in its Office 365 applications. With tabs, groups, numerous supported group items, toolbars, backstage view and above all, the simplified view, Ribbon lets you create latest Microsoft Office 2019 style applications with minimal code and great functionality. One can easily categorize the related menu items into tabs and then groups for that organized and clean user interface. Support for contextual tab group gives you further flexibility to show some tabs only when they are required. Gallery, variety of light and dark themes, Quick Access Toolbar (QAT) and icon sets are some other essential features to enhance the end-user experience. Further more, the full-fledged API of **Ribbon for WinForms** provides all these features without any compromise with the run-time performance of your application.

Note that the new Ribbon for WinForms is a better, faster and latest version of the already existing Ribbon for WinForms, now referred to as Classic Ribbon for WinForms. Existing users of Classic Ribbon may migrate to the latest version by following some simple steps described in Upgrade from Classic Ribbon to New Ribbon topic.



クラシックリボンから新しいリボンへのアップグレード

A user can upgrade from classic Ribbon to the new Ribbon without any hassles. The image below depicts the appearance of the old Ribbon control at runtime.



This section covers the steps required to upgrade the old ribbon control to new Ribbon control:

Step 1: Make Reserve Copy

Make a reserve copy of your old project. You might need it during conversion if something doesn't get right.

Step 2: Update Project

Remove the C1.Win.C1Ribbon.4 assembly from References in the Solution Explorer, and add the **C1.Win.Ribbon.4.5.2** assembly. Locate the licenses.licx file in your project and replace all the lines that refer to C1.Win.C1Ribbon.4 with C1.Win.Ribbon.4.5.2. Also, replace all C1.Win.C1Ribbon namespace with **C1.Win.Ribbon**.

Step 3: Handle API Changes

Build the project. The user can observe that the designer has used **IconSet** property in place of Image, **SmallImage** and **LargeImage** properties. You can also see the error list with obsolete API. The changes have been made intentionally and indicate the following API changes:

- To fix the errors related to **RibbonEventHandler**, change the subscription on RibbonEvent as the RibbonEventHandler has been removed.

  ```
  this.c1Ribbon1.RibbonEvent += new RibbonEventHandler(c1Ribbon1_RibbonEvent);
  ```

  to

  ```
  this.c1Ribbon1.RibbonEvent += c1Ribbon1_RibbonEvent;
  ```

- To fix the errors related to AppMenuAppearance enum, remove related code lines. Note that the **C1BackstageView** is a separate component now, so application can't setup it via the AppMenuAppearance enumeration member.
- To fix the errors related to **VisualStyle** and **VisualStyleHolder** properties and **ResetVisualStyle** method, remove all the related code lines as the Ribbon Form does not anymore contain such property or method.
- To fix the errors related to **VerticalLayout** property from the RibbonGalleryItem class, remove all related lines of code and use the **GalleryItemTextImageRelation** property.

Step 4: Use IconSet property

If you open your form with Ribbon in Visual Studio designer, then you can observe that the Image, SmallImage and LargeImage properties have changed to IconSet property. You can see that instead of image-related properties, code lines like these have appeared:

```
this.FontBoldButton.IconSet.Add(new C1.Framework.C1BitmapIcon("", new
System.Drawing.Size(16, 16), System.Drawing.Color.Transparent,
((System.Drawing.Image)(resources.GetObject("FontBoldButton.IconSet")))));
```

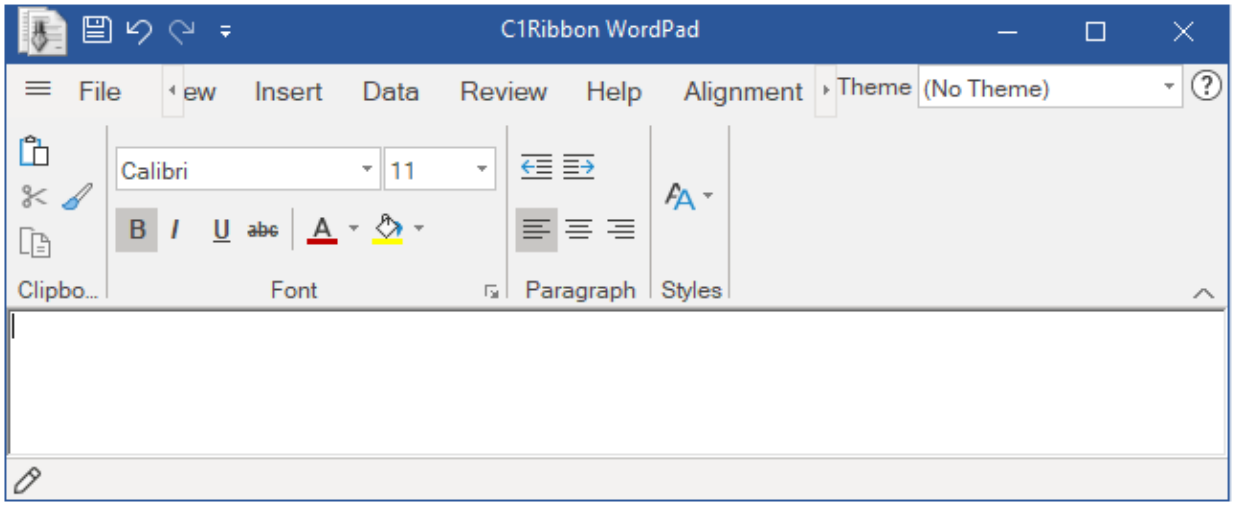
Note that the new Ribbon has an updated set of embedded images. If you want to use them, you can replace the images for individual elements using the Smart Designer. You can replace all the old images from embedded preset with new ones. Not only do they look better, it is also useful while switching to a simplified view. For example, retaining the old IncreaseIndentButton image in the new ribbon may give it a blurry look in the simplified mode. Hence, it is preferred to replace all old images.

Note: The new Ribbon components have **Image**, **SmallImage** and **LargeImage** properties. These properties are used for compatibility with the classic version of the C1Ribbon, and can't be serialized and are hidden from the PropertyGrid.

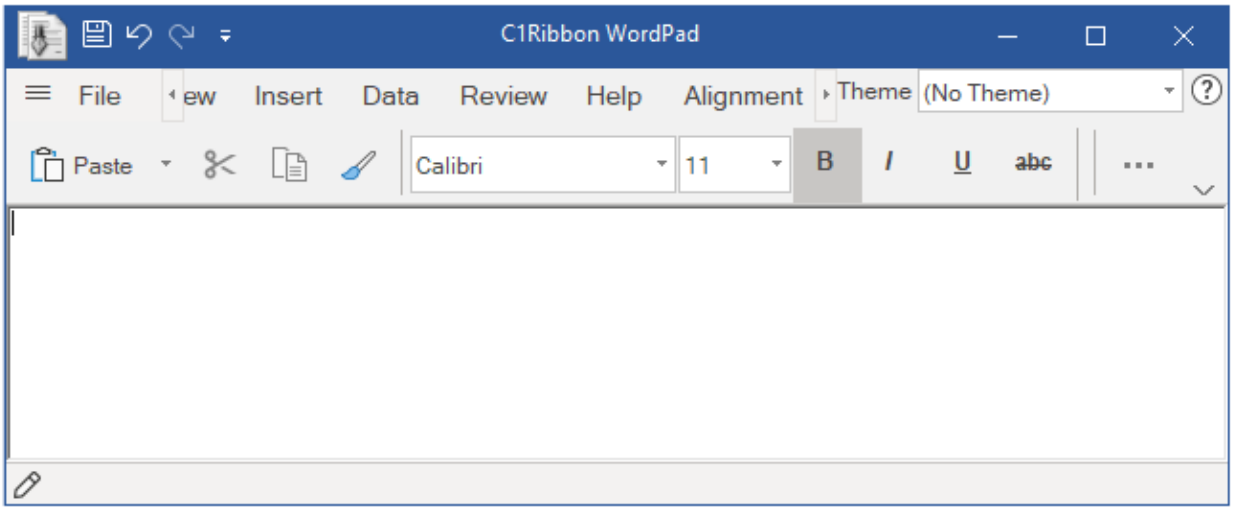
Step 5: Run the application

Run your application and make sure that all the functionalities are working as expected.

The upgraded Ribbon control now looks like this:



The new **C1Ribbon** supports a feature called the Simplified Ribbon. The end-user can switch between the Simplified and Full views using the chevron button in the bottom right corner of the ribbon. This is how the ribbon looks in its simplified view:



主な特長

The major features of **C1Ribbon** control are elucidated below:

- **Simplified Ribbon**

  Ribbon for WinForms is a simplified compact ribbon, which is streamlined to provide more space for content. It enables a user to toggle back and forth between the simplified and full view at runtime by clicking the chevron button located towards the lower right corner of the ribbon. The Simplified View allows the display of some commands in a single line, and others accessible through a drop-down menu.

- **Backstage**

  C1BackstageView is a separate component in **C1Ribbon** that imitates the **Main Application Menu** in old Ribbon-based applications. The backstage opens in a full-size window and works like a multi-level navigation control. A user can access the backstage by clicking the main button located on the leftmost corner of the application window. It replaces the main application menu button. The backstage provides the user access to functions like opening, saving, creating, renaming and printing files.

- **Gallery**

  The **Gallery** Bar provides a gallery of items that you can select by clicking them. **C1Ribbon** supports some interesting features like grouping, filtering and zooming of items within the gallery.

- **Quick Access Toolbar**

  The **Quick Access Toolbar** (QAT) provides end-users the access to frequently-used items, such as New, Open, Save, Undo, Redo etc. The user can customize the Quick Access Toolbar and add more items from tabs and groups in the ribbon.

- **Icons**

  The Ribbon uses icon sets to display items with different sizes in various display modes and screen resolutions. The classic ribbon control offered only two options, large (32x32) and small (16x16) preset images for the Ribbon items, but the new **C1Ribbon** control defines appropriate icon sizes for low, medium and high DPI environments.

- **New Images in Themes**

  The **C1Ribbon** control includes a new set of embedded images for both light (low contrast) and dark (high contrast) themes. This helps to confer a better visual appearance to the ribbon.

- **Key Tips**

  The **C1Ribbon** control supports KeyTips to help the user reach a particular Ribbon command more easily by pressing a keyboard letter or number instead of mouse clicks.

ComponentOneテンプレートの使用

ComponentOne WinForms Edition provides predefined project templates to create and configure WinForms Ribbon applications in Visual Studio. These project templates get registered in Visual Studio after the ComponentOne Studio installer is run to install WinForms Edition. The template provides several benefits over Visual Studio templates as it automatically adds the required resources to the project, as listed below:

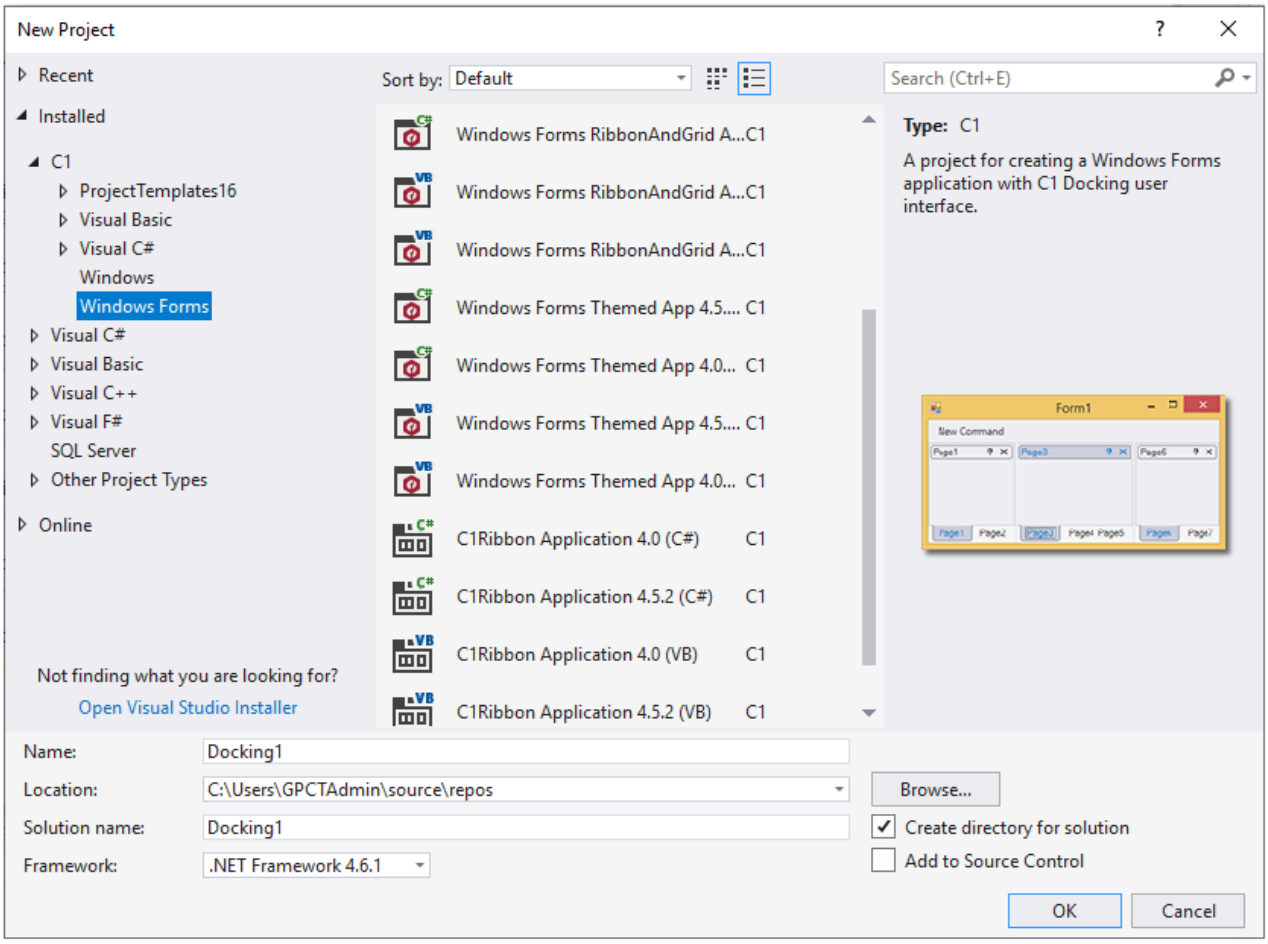
- Adds required WinForms Edition references to the project.
- Adds Licensing.licx file containing valid licenses to the project.
- Adds a sample to see the related control or component in action.

C1 Windows Forms project templates for **C1.Win.Ribbon** are available in both, C# and VB programming languages.

- C1Ribbon Application 4.0 (C#)
- C1Ribbon Application 4.5.2 (C#)
- C1Ribbon Application 4.0 (VB)
- C1Ribbon Application 4.5.2 (VB)

Complete the following steps to create a WinForms project in Visual Studio using ComponentOne project templates:

1. Open Visual Studio.
2. Click **File** menu and select **New | Project**. The **New Project** dialog appears.
3. Under **Installed** in the left pane, click **C1** to see the available programming languages, Visual Basic and Visual C#.
4. Choose a language.
5. Under the selected language, choose **Windows Forms** to see a list of available ComponentOne templates for Ribbon in the center pane.



6. Select the required template.
7. Type project name, set its location, and click **OK** to create a new WinForms project.

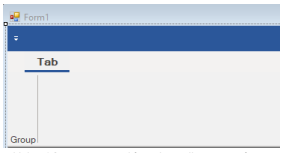
クイックスタート

The following quick start guide will take you through the basics of Ribbon control and its design customizations. You can add tabs, groups and items to the groups. By the end of this section, you will be able to create a simple text editor with the Ribbon user interface.

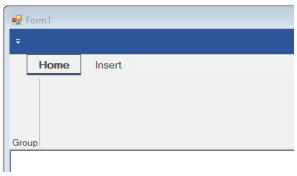
.NET 4.5.2

To create a simple WinForms application in .NET 4.5.2 for Ribbon control, complete the following steps:

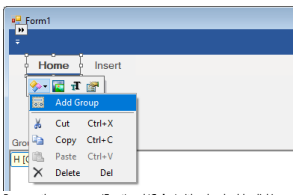
1. Create a new **Windows Forms** application.
2. Drag and drop the **C1Ribbon** control to your form.The Ribbon control is positioned at the top of the form. A tab and group gets added to the Ribbon control in advance.



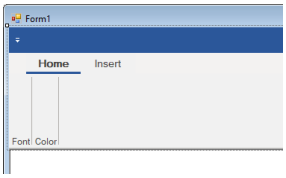
3. Add the **RichTextBox** control from the Toolbox to your form. Set its **Dock** property to **Fill.**
4. To add a new tab, click the **C1Ribbon** control to enable its floating toolbar. Click the **Actions** button and select **Add Tab** from the list of actions. Whenever you add a new tab, a group is already added to it.
5. Rename the tabs as '**Home**' and '**Insert**'. This can be done by in-place text editing by double clicking the tab. You can also add text from the **Text Settings** in floating toolbar or by editing the **Text** property in the Properties window.



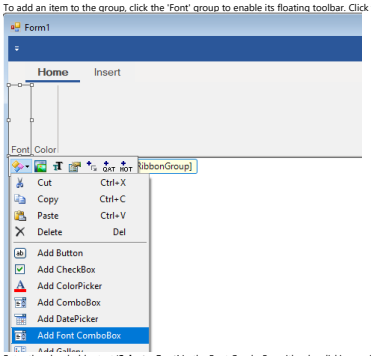
6. To add a new group, click the 'Home' Tab to enable its floating toolbar. Click the **Actions** button and select **Add Group** from the list of actions.



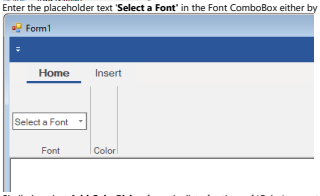
7. Rename the groups as '**Font**' and '**Color**' either by double clicking on it or by editing the **Text** property from the **Properties** window.



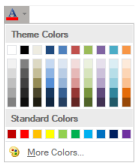
8. To add an item to the group, click the 'Font' group to enable its floating toolbar. Click the Actions button and Select **Add Font ComboBox** from the list of actions.



9. Enter the placeholder text '**Select a Font**' in the Font ComboBox either by clicking on it or editing the **Text** property from the **Properties** window.



10. Similarly, select **Add ColorPicker** from the list of actions of 'Color' group to add it as an item.



11. The Windows Form can be converted to a Ribbon Form by switching to the code view and replacing the following:

C# copyCode

```
partial class Form1 : Form
{
    //...
}
```

VB copyCode

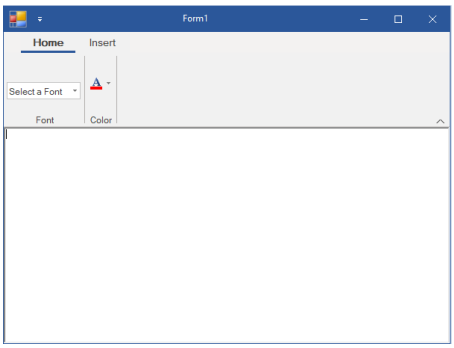
```
Partial Class Form1
    Inherits System.Windows.Forms.Form
    '...
End Class
```

With

C# copyCode
```
partial class Form1 : C1RibbonForm
{
    //...
}
```
VB copyCode
```
Partial Class Form1
    Inherits C1.Win.C1Ribbon.C1RibbonForm
    '...
End Class
```

The basic difference between the Windows form and the Ribbon Form is that in the Ribbon form, the title bar of the Windows Form is omitted and the title bar of the Ribbon Form comes into existence.

Run the application and observe the Ribbon Form generated in the following image. The Ribbon Form contains tabs, groups and FontComboBox and ColorPicker as items in the Group. Similarly, you can add other items from the group item list such as DatePicker, Gallery, Label, CheckBox, ToggleButton, TextBox etc.



.NET 5

To create a simple WinForms application in .NET 5 for Ribbon control, complete the following steps:

1. Create a new **Windows Forms** application.
2. Switch to code editor and initialize the C1Ribbon object.

C# copyCode
```
C1Ribbon customRibbon;
```

3. Initialize the Ribbon control and add the control to the Form.

C# copyCode
```
//  Ribbonコントロールを初期化します
customRibbon = new C1Ribbon();
//  Ribbonコントロールをフォームに追加します
this.Controls.Add(customRibbon);
```

4. Create an instance of RibbonTab class and add the 'Home' tab to the Ribbon control using the **Add** method of RibbonTabCollection class.

C# copyCode
```
//「ホーム」タブを作成して追加します
RibbonTab homeTab = new RibbonTab();
homeTab.Text = "Home";
customRibbon.Tabs.Add(homeTab);
```

5. Create an instance of RibbonGroup class, assign it the name 'Font' and add it to the Home tab using the Add method of RibbonGroupCollection class.

C# copyCode
```
//「ホーム」タブに「フォント」という名前のグループを追加します
RibbonGroup fontStyle = new RibbonGroup();
fontStyle.Text = "Font";
homeTab.Groups.Add(fontStyle);
```

6. Create Quick Access Toolbar in the Ribbon control using the Qat property of C1Ribbon class, and add items to the Quick Access Toolbar using the Items property of RibbonQat class and Add method of RibbonItemCollectionBase class.

C# copyCode
```
// クイックアクセスツールバーにアイテムを追加します
Image openImage = Image.FromFile(@"images\open.png");
customRibbon.Qat.Items.Add(new RibbonButton("Open", openImage));
Image undoImage = Image.FromFile(@"images\undo.png");
customRibbon.Qat.Items.Add(new RibbonButton("Undo", undoImage));
Image redoImage = Image.FromFile(@"images\redo.png");
customRibbon.Qat.Items.Add(new RibbonButton("Repeat", redoImage));
```

7. Create items to be added in application menu. Add them to the right and left panel of the Application Menu using **ApplicationMenu** property of C1Ribbon class.

C# copyCode
```
// デフォルトのアプリケーションメニューに追加する項目を作成します
RibbonButton openButton = new RibbonButton("Open", Image.FromFile(@"images\open-file-icon.png"));
RibbonButton saveButton = new RibbonButton("Save", Image.FromFile(@"images\save-file-icon.png"));
RibbonButton closeButton = new RibbonButton("Close", Image.FromFile(@"images\close.png"));
RibbonButton printButton = new RibbonButton("Print", Image.FromFile(@"images\print.png"));
RibbonButton previewButton = new RibbonButton("Preview", Image.FromFile(@"images\preview.png"));
RibbonListItem print = new RibbonListItem(printButton);
RibbonListItem preview = new RibbonListItem(previewButton);
// ApplicationMenuの画像アイコンを追加します
C1BitmapIcon c1Bitmap1 = new C1BitmapIcon();
c1Bitmap1.Source = Image.FromFile(@"images\application_menu.png");
c1Bitmap1.Size = new Size(20, 20);
customRibbon.ApplicationMenu.IconSet.Add(c1Bitmap1);
// ApplicationMenuの項目を追加します
customRibbon.ApplicationMenu.LeftPaneItems.Add(openButton);
customRibbon.ApplicationMenu.LeftPaneItems.Add(saveButton);
customRibbon.ApplicationMenu.LeftPaneItems.Add(closeButton);
customRibbon.ApplicationMenu.RightPaneItems.Add(print);
customRibbon.ApplicationMenu.RightPaneItems.Add(preview);
```

8. Create items to be added in configuration toolbar. Add the items using ConfigToolBar property of C1Ribbon class.

C# copyCode
```
// デフォルトのConfigToolbarに追加する項目を作成します
RibbonButton cutButton = new RibbonButton("Cut", Image.FromFile(@"images\cut.png"));
RibbonButton copyButton = new RibbonButton("Copy", Image.FromFile(@"images\copy.png"));
RibbonButton pasteButton = new RibbonButton("Paste", Image.FromFile(@"images\paste.png"));
// ConfigToolbarの項目を追加します
customRibbon.ConfigToolBar.Items.Add(cutButton);
customRibbon.ConfigToolBar.Items.Add(copyButton);
customRibbon.ConfigToolBar.Items.Add(pasteButton);
```

9. Add a new tab 'Format' and new 'Clipboard' group. Add items like button, checkbox, fontcombobox, colorpicker, toggle button and separator to the new tab and group.

C# copyCode
```
// ボタン、チェックボックス、fontcombobox、colorpicker、トグルボタン、セパレーターなどの項目を追加します
// 新しいタブを作成します
RibbonTab format = new RibbonTab();
format.Text = "Format";
customRibbon.Tabs.Add(format);
// グループを追加します
RibbonGroup formatGroup = new RibbonGroup();
formatGroup.Text = "Clipboard";
```

```
format.Groups.Add(formatGroup);
// リボンボタンを追加します
RibbonButton clearButton = new RibbonButton("Clear Format", Image.FromFile(@"images\clearformat.png"));
clearButton.ToolTip = "Clear All Formatting";
formatGroup.Items.Add(clearButton);
// チェックボックスボタンのリボン項目を追加します
RibbonCheckBox reminderCheckBox = new RibbonCheckBox();
reminderCheckBox.IconSet.Add(new C1BitmapIcon(null, new Size(20, 20), Color.Transparent, Image.FromFile(@"images\reminder.png")));
reminderCheckBox.Text = "Reminder";
formatGroup.Items.Add(reminderCheckBox);
// RibbonFontComboBoxを追加します
RibbonFontComboBox fontComboBox = new RibbonFontComboBox();
fontComboBox.ToolTip = "Font";
fontComboBox.Text = "Select a font";
formatGroup.Items.Add(fontComboBox);
// RibbonSeparatorを追加します
RibbonSeparator separator = new RibbonSeparator();
formatGroup.Items.Add(separator);
// RibbonColorPickerを追加します
RibbonColorPicker colorPicker = new RibbonColorPicker(null, Image.FromFile(@"images\fontcolor.png"));
colorPicker.ToolTip = "Color Picker";
formatGroup.Items.Add(colorPicker);
// トグルボタンを追加します
RibbonToggleButton leftAlign = new RibbonToggleButton(Image.FromFile(@"images\align_left.png"));
leftAlign.ToolTip = "Align your content with left margin.";
formatGroup.Items.Add(leftAlign);
```

10. Add items like datepicker, gallery, label and menu to another tab and group.

C# copyCode

```
// 日付ピッカー、ギャラリー、ラベル、メニューなどの項目を追加します
// 新しいタブ「ツール」を作成します
RibbonTab tool = new RibbonTab();
tool.Text = "Tools";
customRibbon.Tabs.Add(tool);
// 「ツール」タブにエディターグループを追加します
RibbonGroup toolGroup = new RibbonGroup();
toolGroup.Text = "Editor";
tool.Groups.Add(toolGroup);
// DatePickerリボン項目を追加します
RibbonDatePicker datePicker = new RibbonDatePicker();
datePicker.Format = "yyyy/MM/dd";
toolGroup.Items.Add(datePicker);
// RibbonGalleryリボン項目を追加します
RibbonGallery ribbonGallery = new RibbonGallery();
ribbonGallery.ToolTip = "Select a shape";
RibbonGalleryItem ribbonGallery1 = new RibbonGalleryItem("Rectangle", Image.FromFile(@"images\rect.png"));
RibbonGalleryItem ribbonGallery2 = new RibbonGalleryItem("Circle", Image.FromFile(@"images\circle.png"));
RibbonGalleryItem ribbonGallery3 = new RibbonGalleryItem("Triangle", Image.FromFile(@"images\triangle.png"));
RibbonGalleryItem ribbonGallery4 = new RibbonGalleryItem("Hexagon", Image.FromFile(@"images\hexagon.png"));
ribbonGallery.Items.Add(ribbonGallery1);
ribbonGallery.Items.Add(ribbonGallery2);
ribbonGallery.Items.Add(ribbonGallery3);
ribbonGallery.Items.Add(ribbonGallery4);
toolGroup.Items.Add(ribbonGallery);
// ラベルリボン項目を追加します
RibbonLabel ribbonLabel = new RibbonLabel("Feedback", Image.FromFile(@"images\feedback.png"));
formatGroup.Items.Add(ribbonLabel);
// RibbonMenuリボン項目を追加します
RibbonMenu ribbonMenu = new RibbonMenu();
ribbonMenu.Text = "Edit";
RibbonButton buttonCut = new RibbonButton("Cut", Image.FromFile(@"images\cut.png"));
RibbonButton buttonCopy = new RibbonButton("Copy", Image.FromFile(@"images\copy.png"));
RibbonButton buttonPaste = new RibbonButton("Paste", Image.FromFile(@"images\paste.png"));
RibbonSeparator separatoritem = new RibbonSeparator();
RibbonColorPicker colorpicker = new RibbonColorPicker("Color Picker", Image.FromFile(@"images\fontcolor.png"));
RibbonLabel label = new RibbonLabel("Select Color");
RibbonColorPickerItem colorItem = new RibbonColorPickerItem();
ribbonMenu.Items.Add(buttonCut);
ribbonMenu.Items.Add(buttonCopy);
ribbonMenu.Items.Add(buttonPaste);
ribbonMenu.Items.Add(separatoritem);
ribbonMenu.Items.Add(colorpicker);
ribbonMenu.Items.Add(label);
ribbonMenu.Items.Add(colorItem);
toolGroup.Items.Add(ribbonMenu);
```

11. Add more items like Numeric box, progress bar, splitbutton, textbox, timepicker, toolbar and trackbar.

C# copyCode

```
// Add more items like Numeric box, progress bar, splitbutton, textbox, timepicker, toolbar and trackbar
// Add Styles group to Tools tab
RibbonGroup stylesGroup = new RibbonGroup();
stylesGroup.Text = "Styles";
tool.Groups.Add(stylesGroup);
// Add Numeric Box ribbon item
RibbonNumericBox numericBox = new RibbonNumericBox();
numericBox.Maximum = 10;
numericBox.Minimum = 1;
numericBox.ToolTip = "Select a number";
numericBox.Label = "Number Box";
stylesGroup.Items.Add(numericBox);
// Add Progress Bar ribbon item
RibbonProgressBar progressBar = new RibbonProgressBar();
progressBar.Minimum = 0;
progressBar.Maximum = 100;
progressBar.Value = 30;
stylesGroup.Items.Add(progressBar);
// Add Separator ribbon item
RibbonSeparator separatorItem = new RibbonSeparator();
stylesGroup.Items.Add(separatorItem);
// Add Ribbon Split Button
RibbonSplitButton splitButton = new RibbonSplitButton("Views");
RibbonButton printLayout = new RibbonButton("Print Layout", Image.FromFile(@"images\printlayout.png"));
RibbonButton fullScreenLayout = new RibbonButton("Full Screen Reading", Image.FromFile(@"images\fullscreen.png"));
RibbonButton webLayout = new RibbonButton("Web Layout", Image.FromFile(@"images\weblayout.png"));
splitButton.Items.Add(printLayout);
splitButton.Items.Add(fullScreenLayout);
splitButton.Items.Add(webLayout);
stylesGroup.Items.Add(splitButton);
// Add Ribbon TextBox
RibbonTextBox textBox = new RibbonTextBox();
textBox.Text = "Write Text";
stylesGroup.Items.Add(textBox);
// Add TimePicker
RibbonTimePicker timePicker = new RibbonTimePicker();
timePicker.Label = "Select Time:";
stylesGroup.Items.Add(timePicker);
// Add Ribbon Toolbar
RibbonToolBar ribbonToolBar = new RibbonToolBar();
RibbonButton boldButton = new RibbonButton(Image.FromFile(@"images\bold.png"));
RibbonButton italicButton = new RibbonButton(Image.FromFile(@"images\italic.png"));
RibbonButton underlineButton = new RibbonButton(Image.FromFile(@"images\underline.png"));
ribbonToolBar.Items.Add(boldButton);
ribbonToolBar.Items.Add(italicButton);
ribbonToolBar.Items.Add(underlineButton);
stylesGroup.Items.Add(ribbonToolBar);
// Add TrackBar
RibbonTrackBar trackBar = new RibbonTrackBar();
trackBar.Minimum = 10;
trackBar.Maximum = 100;
trackBar.Value = 30;
trackBar.TickFrequency = 10;
stylesGroup.Items.Add(trackBar);
```

Note that the Ribbon control provides dedicated class for each item in the ribbon, such as RibbonButton, RibbonCheckBox, RibbonFontComboBox, RibbonSeparator, RibbonColorPicker, RibbonToggleButton, RibbonDatePicker, RibbonGallery, RibbonLabel, RibbonMenu, RibbonNumericBox, RibbonProgressBar, RibbonSplitButton, RibbonTextBox, RibbonTimePicker, RibbonToolbar, RibbonTrackBar classes.

12. Ribbon control also allows the user to add a hosted control in the ribbon using the RibbonControlHost class. For this, the user has to create a new TextBoxHost class that inherits the RibbonControlHost element, and then initialize RibbonControlHost object in the form and add it to the ribbon group.

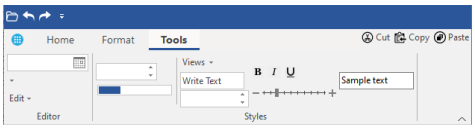
C# copyCode

```
// コントロールホストを追加します
RibbonControlHost textboxHost = new RibbonControlHost();
textboxHost = new WinFormsRibbonQuickStart.TextBoxHost();
stylesGroup.Items.Add(textboxHost);
```

C# copyCode

```
// 新しいTextBoxHostクラスを作成します
public class TextBoxHost : C1.Win.Ribbon.RibbonControlHost
{
    public TextBoxHost() : base(new System.Windows.Forms.TextBox())
    {
        base.Text = "Sample text";
    }
}
```

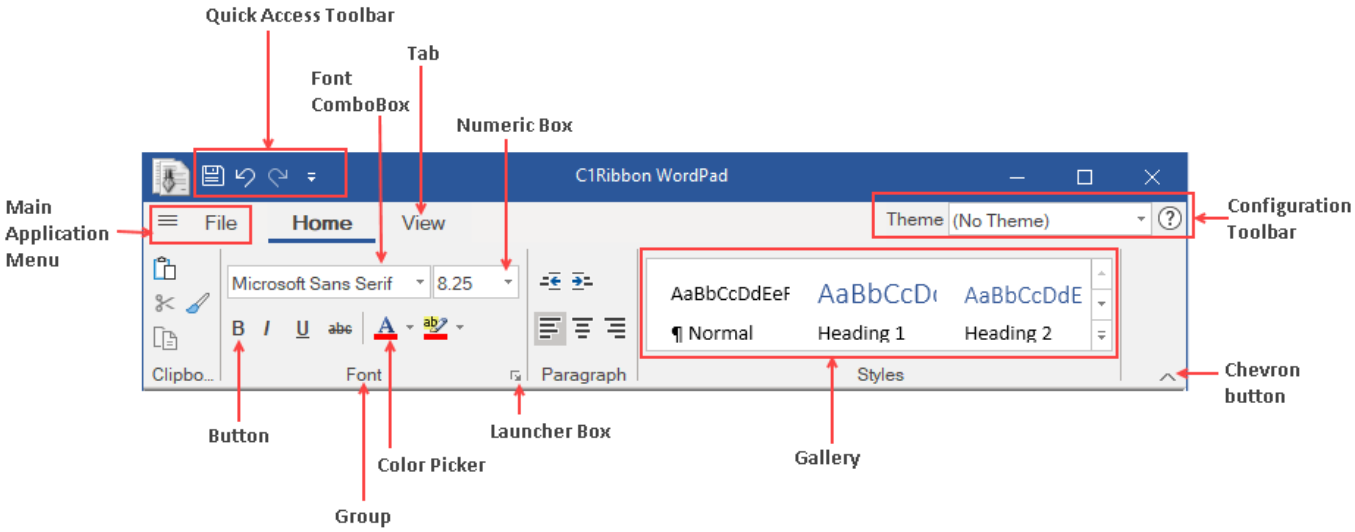
Run the application and observe the ribbon control with tabs, groups and ribbon items.



Note: WinForms .NET 5 Edition does not include rich design-time support yet. We will enhance it in future releases.

要素

C1Ribbon comprises several visual elements, each of which represents a different functionality. The image below depicts some of the UI elements provided by the Ribbon control:



The following topics throw more light on the UI elements in **C1Ribbon**.

Application Menu

BackStageView

Quick Access Toolbar

Configuration Toolbar

Contextual Tab Group

Ribbon Tab

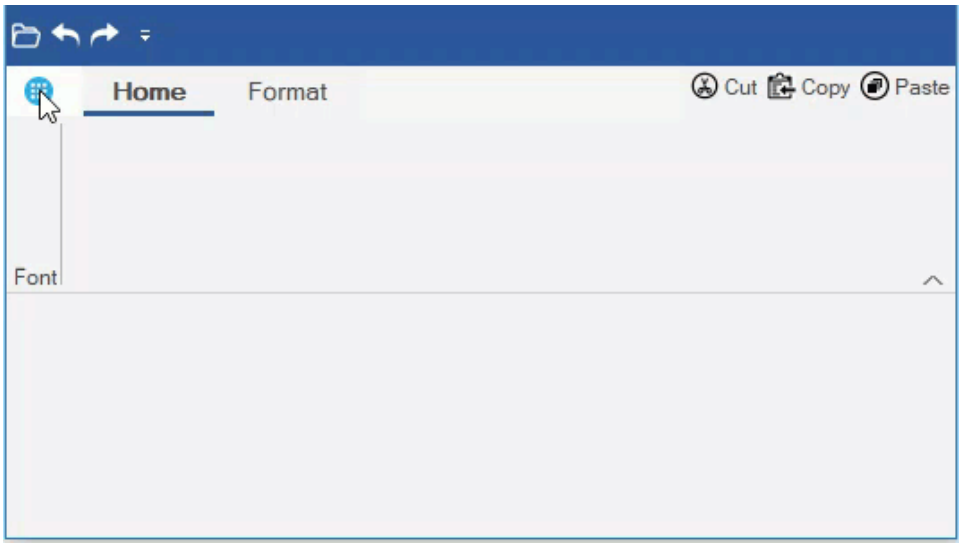
Ribbon Item Group

Ribbon Items

アプリケーション メニュー

The **Application Menu** is located towards the top-left corner of the ribbon control. This application menu button when clicked displays a dropdown menu of specific commands that can be performed on the entire application such as **Open**, **Close**, **Save**, **Save As**, **Print** etc.

The GIF below shows the UI of the Application Menu.

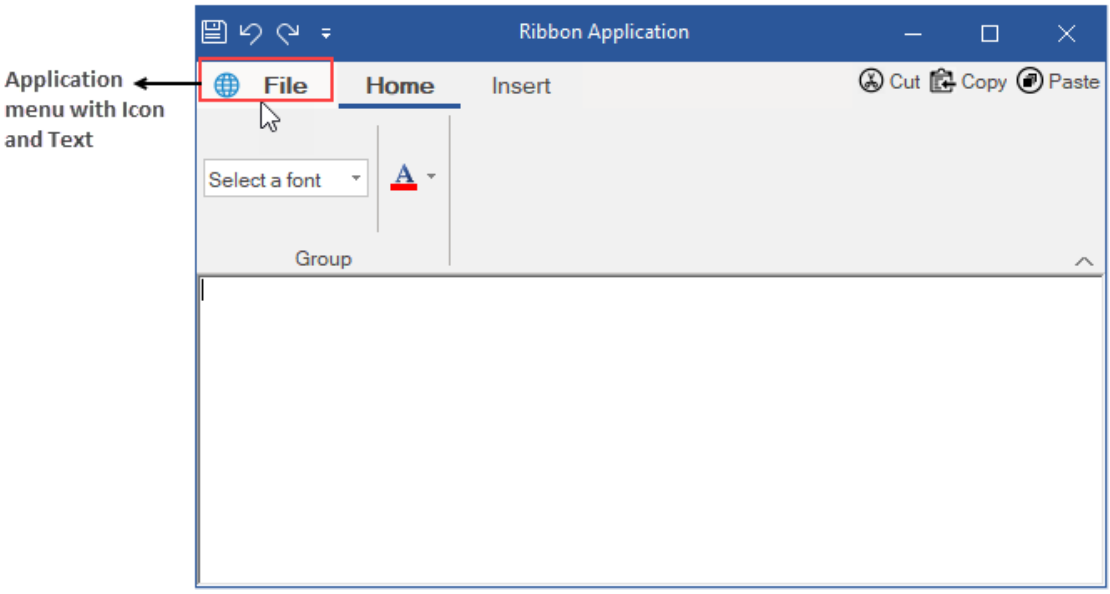


The following section discusses configuring Application Menu and adding items to the drop-down.

Configuring the Application Menu button

The user may note that when the **C1Ribbon** control is dropped on the **Form**, it contains by default an Application Menu button, a tab and group. This Application Menu button can be displayed using text, image, or a combination of both.

The image below shows a ribbon with Application Menu button containing both icon and text.



The text and image can be changed at design time through the Application Menu's floating toolbar or by setting the **Text** and **IconSet** properties in the **Properties** Window. For more information, refer this topic.

Programmatically, the Application Menu button can be configured using the **ApplicationMenu ('ApplicationMenu プロパティ' in the on-line documentation)** property of **C1Ribbon ('C1Ribbon クラス' in the on-line documentation)** class and the **IconSet ('IconSet プロパティ' in the on-line documentation)** property of **RibbonIconItem ('RibbonIconItem クラス' in the on-line documentation)** class. The icon of the menu button can be configured using the Source property of C1BitmapIcon class and Size property of C1Icon class.

This is depicted in the code snippet below:

- **Visual Basic**

```
' ApplicationMenuの画像アイコンを追加します
```

```
Dim c1Bitmap1 As New C1BitmapIcon()
    c1Bitmap1.Source = Image.FromFile("images\app_menu.png")
    c1Bitmap1.Size = New Size(20, 20)
    customRibbon.ApplicationMenu.IconSet.Add(c1Bitmap1)
```

- **C#**

```
//ApplicationMenuの画像アイコンを追加します
C1BitmapIcon c1Bitmap1 = new C1BitmapIcon();
c1Bitmap1.Source = Image.FromFile(@"images\application_menu.png");
c1Bitmap1.Size = new Size(20, 20);
customRibbon.ApplicationMenu.IconSet.Add(c1Bitmap1);
```

Adding Items to Application Menu drop-down

The commands in the Application Menu dropdown can be customized through the designer as well as code. They can be arranged on the menu using **Floating Toolbar** and **Collection Editors** at design-time. This lets the user arrange the commands in the left, right or bottom pane of the Application Menu. For more information about using Collection Editors, refer this topic**.**

The user can also add commands in the Application Menu through code. This can be done by creating new instances of the ribbon buttons, using the **RibbonButton ('RibbonButton クラス' in the on-line documentation)** class. These buttons can then be added to the menu dropdown using **LeftPaneItems ('LeftPaneItems プロパティ' in the on-line documentation)**, **RightPaneItems ('RightPaneItems プロパティ' in the on-line documentation)** and **BottomPaneItems ('BottomPaneItems プロパティ' in the on-line documentation)** properties of the **RibbonApplicationMenu ('RibbonApplicationMenu クラス' in the on-line documentation)** class.

This is depicted in the code snippet below:

- **Visual Basic**

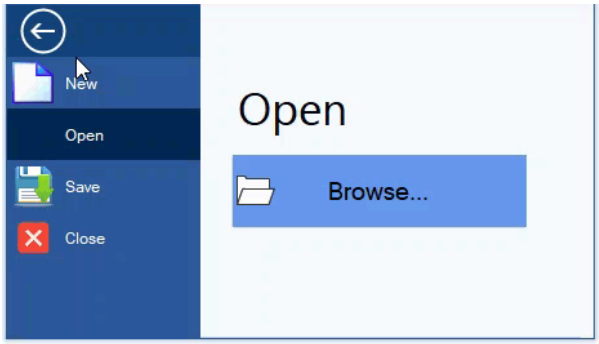
```
'デフォルトのアプリケーションメニューに追加するアイテムを作成します
Dim openButton As New RibbonButton("Open", Image.FromFile("images\open-file-icon.png"))
Dim saveButton As New RibbonButton("Save", Image.FromFile("images\save-file-icon.png"))
Dim closeButton As New RibbonButton("Close", Image.FromFile("images\close.png"))
Dim printButton As New RibbonButton("Print", Image.FromFile("images\print.png"))
Dim previewButton As New RibbonButton("Preview", Image.FromFile("images\preview.png"))

Dim print As New RibbonListItem(printButton)
Dim preview As New RibbonListItem(previewButton)

'アプリケーションメニューのアイテムを追加します
customRibbon.ApplicationMenu.LeftPaneItems.Add(openButton)
customRibbon.ApplicationMenu.LeftPaneItems.Add(saveButton)
customRibbon.ApplicationMenu.LeftPaneItems.Add(closeButton)
customRibbon.ApplicationMenu.RightPaneItems.Add(print)
customRibbon.ApplicationMenu.RightPaneItems.Add(preview)
```

- **C#**

```
//デフォルトのアプリケーションメニューに追加するアイテムを作成します
RibbonButton openButton = new RibbonButton("Open", Image.FromFile(@"images\open-file-icon.png"));
RibbonButton saveButton = new RibbonButton("Save", Image.FromFile(@"images\save-file-icon.png"));
RibbonButton closeButton = new RibbonButton("Close", Image.FromFile(@"images\close.png"));
RibbonButton printButton = new RibbonButton("Print", Image.FromFile(@"images\print.png"));
RibbonButton previewButton = new RibbonButton("Preview", Image.FromFile(@"images\preview.png"));
RibbonListItem print = new RibbonListItem(printButton);
RibbonListItem preview = new RibbonListItem(previewButton);

//アプリケーションメニューのアイテムを追加します
customRibbon.ApplicationMenu.LeftPaneItems.Add(openButton);
customRibbon.ApplicationMenu.LeftPaneItems.Add(saveButton);
customRibbon.ApplicationMenu.LeftPaneItems.Add(closeButton);
customRibbon.ApplicationMenu.RightPaneItems.Add(print);
customRibbon.ApplicationMenu.RightPaneItems.Add(preview);
```

Backstage ビュー

Backstage View is a feature that has been a part of Office applications since the 2007 version. This component replaces the **Application Menu**, conventionally called the **File** tab. In the new Ribbon control, the Backstage appears as the main button at the top-left corner, which when clicked opens a full-size backstage window. It contains the functionality previously found in the File menu, such as Open, Save, Save As, New, Print etc.



The **C1BackstageView** is a separate component and can be added at design-time from the Toolbox. The **C1Ribbon** control can be integrated with **C1BackstageView** component through the designer as well as code. The Backstage window contains the left and left bottom panes. The user can add items to both the panes in the window.

Backstage can be configured both by the designer and code. For design-time configuration, refer this topic.

The **BackstageView** Tab when clicked should open a collection of User Controls at runtime. This can be configured only through the code by creating instances of User Controls and assigning them to the **Control ('Control プロパティ' in the on-line documentation)** property of the **BackstageViewTab ('BackstageViewTab クラス' in the on-line documentation)** class. This is depicted in the code below.

- **Visual Basic**

```
Public Class BackstageView
    Public Property Owner() As C1BackstageView
        Get
            Return m_Owner
        End Get
        Friend Set
            m_Owner = Value
        End Set
    End Property

    Private m_Owner As C1BackstageView

    Private Sub browseCaption_Click_1(sender As Object, e As EventArgs) Handles browseCaption.Click
        Dim openDialog As OpenFileDialog
        openDialog = New OpenFileDialog()
        openDialog.Title = "Browse .rtf file"
        openDialog.Filter = "Rich Text Files (*.rtf)|*.RTF"

        If openDialog.ShowDialog() = DialogResult.OK Then
            Dim rct As RichTextBox = DirectCast(DirectCast(sender, Control).TopLevelControl.Controls(1), RichTextBox)
            rct.LoadFile(openDialog.FileName)
            Owner.DroppedDown = False
        End If
    End Sub

    Private Sub browseCaption_MouseHover(sender As Object, e As EventArgs) Handles browseCaption.MouseHover
        browseCaption.Font = New Font(browseCaption.Font.Name, browseCaption.Font.SizeInPoints, FontStyle.Underline)
        browseCaption.BackColor = Color.AliceBlue
    End Sub

    Private Sub browseCaption_MouseLeave(sender As Object, e As EventArgs) Handles browseCaption.MouseLeave
        browseCaption.BackColor = Color.CornflowerBlue
        browseCaption.Font = New Font(browseCaption.Font.Name, browseCaption.Font.SizeInPoints, FontStyle.Regular)
    End Sub
End Class
```

- **C#**

```
public BackstageOpenView()
{
    InitializeComponent();
}
public C1BackstageView Owner { get; internal set; }

private void browseCaption_Click(object sender, EventArgs e)
{
    OpenFileDialog openDialog;
    openDialog = new OpenFileDialog();
    openDialog.Title = "Browse .rtf file";
    openDialog.Filter = "Rich Text Files (*.rtf)|*.RTF";

    if (openDialog.ShowDialog() == DialogResult.OK)
    {
        RichTextBox rct = (RichTextBox)((Control)sender).TopLevelControl.Controls[1];
        rct.LoadFile(openDialog.FileName);
        Owner.DroppedDown = false;
    }
}
private void browseCaption_MouseHover(object sender, EventArgs e)
{
    browseCaption.Font = new Font(browseCaption.Font.Name, browseCaption.Font.SizeInPoints, FontStyle.Underline);
    browseCaption.BackColor = Color.AliceBlue;

}
private void browseCaption_MouseLeave(object sender, EventArgs e)
{
    browseCaption.BackColor = Color.CornflowerBlue;
    browseCaption.Font = new Font(browseCaption.Font.Name, browseCaption.Font.SizeInPoints, FontStyle.Regular);
}
```

Configuring Backstage View By Code

The Backstage View can also be configured through code. The user can create an instance of **C1BackstageView ('C1BackstageView クラス' in the on-line documentation)** class. The Ribbon Buttons can be created using the **RibbonButton ('RibbonButton クラス' in the on-line documentation)** class. Further, the backstage tab can be created using the **BackstageViewTab ('BackstageViewTab クラス' in the on-line documentation)** class, which can be bound to the User Control using the **Control ('Control プロパティ' in the on-line documentation)** property. The buttons and tabs can be added to the BackstageView with the **LeftPaneItems ('LeftPaneItems プロパティ' in the on-line documentation)** property of C1BackstageView class. In order to integrate **C1BackstageView** with the **C1Ribbon** control, you can use the **Owner ('Owner プロパティ' in the on-line documentation)** property of **C1BackstageView** class.

- **Visual Basic**

```
Public Sub New()
    InitializeComponent()
    ApplyBackstageView(Me)
End Sub

Public Sub ApplyBackstageView(form1 As Form1)
    'BackstageViewを作成します
    Dim backstageView As New C1.Win.Ribbon.C1BackstageView()
    backstageView.Text = "File"

    '新しいタブを作成します
    Dim tab = New BackstageViewTab()

    'BackstageViewのユーザーコントロールのインスタンスを作成します
    Dim view As New BackstageView()
    view.Owner = backstageView
    tab.Control = view
    tab.Text = "Open"

    'アイテムをBackStageViewに追加します
    Dim newButton As New RibbonButton("New", Image.FromFile("images\New.png"))
    Dim saveButton As New RibbonButton("Save", Image.FromFile("images\save-file-icon.png"))
    Dim closeButton As New RibbonButton("Close", Image.FromFile("images\close.jpg"))

    'リボンアイテムをBackstageViewに追加します
    backstageView.LeftPaneItems.Add(newButton)

    'タブをBackstageViewに追加します
    backstageView.LeftPaneItems.Add(tab)

    backstageView.LeftPaneItems.Add(saveButton)
    backstageView.LeftPaneItems.Add(closeButton)

    'ボタンに対してエベントイベントハンドラーを追加します
    AddHandler newButton.Click, AddressOf NewButton_Click
    AddHandler saveButton.Click, AddressOf SaveButton_Click
    AddHandler closeButton.Click, AddressOf CloseButton_Click

    'BackstageViewをリボンアイテムに設定します
    backstageView.Owner = C1Ribbon1
    Me.Controls.Add(C1Ribbon1)
End Sub

Private Sub NewButton_Click(sender As Object, e As System.EventArgs)
    Me.RichTextBox1.Clear()
End Sub

Private Sub SaveButton_Click(sender As Object, e As System.EventArgs)
    Dim saveDialog As New SaveFileDialog()
    saveDialog.Title = "Save rich text file"
    saveDialog.Filter = "Rich Text Files (*.rtf)|*.RTF"
    saveDialog.InitialDirectory = "C:\Users\GPCTAdmin\Desktop"
    If saveDialog.ShowDialog() = DialogResult.OK Then
        RichTextBox1.SaveFile(saveDialog.FileName)
    End If
End Sub

Private Sub CloseButton_Click(sender As Object, e As System.EventArgs)
    Me.Close()
End Sub

'RibbonFontComboBoxのSelectedIndexChangedイベントを実装します
Private Sub RibbonFontComboBox1_SelectedIndexChanged(sender As Object, e As EventArgs) Handles RibbonFontComboBox1.SelectedIndexChanged
    RichTextBox1.Font = New Font(RibbonFontComboBox1.Text, RichTextBox1.Font.Size)
End Sub

'ColorPickerのSelectedColorChangedイベントを実装します
Private Sub RibbonColorPicker1_SelectedColorChanged_1(sender As Object, e As EventArgs) Handles RibbonColorPicker1.SelectedColorChanged
    RichTextBox1.SelectionColor = RibbonColorPicker1.Color
End Sub
```

- **C#**

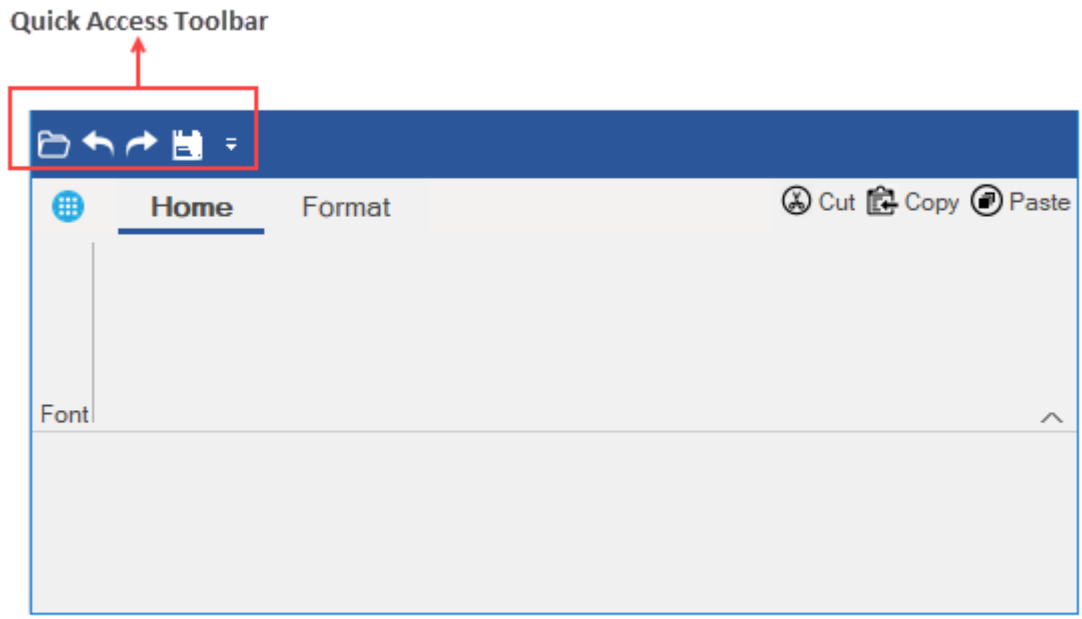
```
public Form1()
{
    InitializeComponent();
    ApplyBackstageView(this);
}

public void ApplyBackstageView(Form1 form1)
{

    //BackstageViewを作成します
    C1.Win.Ribbon.C1BackstageView backstageView = new C1.Win.Ribbon.C1BackstageView();
    backstageView.Text = "File";

    //BackstageViewのユーザーコントロールのインスタンスを作成します
```

```
    BackstageOpenView openView = new BackstageOpenView();
    openView.Owner = backstageView;

    //BackStageViewの新しいタブを作成し、ビューにバインドします
    BackstageViewTab openTab = new BackstageViewTab();
    openTab.Control = openView;
    openTab.Text = "Open";

    //BackStageViewのボタンを作成し、ボタンのイベントハンドラーを定義します
    RibbonButton newButton = new RibbonButton("New", Image.FromFile(@"images\New.png"));
    newButton.Click += NewButton_Click;

    RibbonButton saveButton = new RibbonButton("Save", Image.FromFile(@"images\save-file-icon.png"));
    saveButton.Click += SaveButton_Click;

    RibbonButton closeButton = new RibbonButton("Close", Image.FromFile(@"images\close.jpg"));
    closeButton.Click += CloseButton_Click;

    //リボンアイテム・タブをBackstageViewに追加します
    backstageView.LeftPaneItems.Add(newButton);
    backstageView.LeftPaneItems.Add(openTab);
    backstageView.LeftPaneItems.Add(saveButton);
    backstageView.LeftPaneItems.Add(closeButton);

    //BackstageViewをリボンコントロールに設定します
    backstageView.Owner = customRibbon;

    //リボンコントロールをフォームに追加します
    this.Controls.Add(customRibbon);

}
private void NewButton_Click(object sender, System.EventArgs e)
{
    this.richTextBox1.Clear();
}

private void SaveButton_Click(object sender, System.EventArgs e)
{
    SaveFileDialog saveDialog = new SaveFileDialog();
    saveDialog.Title = "Save rich text file";
    saveDialog.Filter = "Rich Text Files (*.rtf)|*.RTF";
    saveDialog.InitialDirectory = @"C:\Users\GPCTAdmin\Desktop";
    if (saveDialog.ShowDialog() == DialogResult.OK)
    {
        richTextBox1.SaveFile(saveDialog.FileName);
    }
}

private void CloseButton_Click(object sender, System.EventArgs e)
{
    this.Close();
}

//RibbonFontComboBoxのSelectedIndexChangedイベントを実装します
private void ribbonFontComboBox1_SelectedIndexChanged(object sender, System.EventArgs e)
{
    richTextBox1.Font = new Font(ribbonFontComboBox1.Text, richTextBox1.Font.Size);
}

//ColorPickerのSelectedColorChangedイベントを実装します
private void ribbonColorPicker1_SelectedColorChanged(object sender, System.EventArgs e)
{
    richTextBox1.SelectionColor = ribbonColorPicker1.Color;
}
```

クイックアクセスツールバー

The **Quick Access Toolbar** (**QAT**) is a customizable toolbar, which contains a set of quickly accessible commands. It is located on the top left corner of the Ribbon control, and can be displayed up or down the ribbon.

The following image shows a Ribbon with a QAT of four **Ribbon** button items (**Open**, **Undo**, **Redo** and **Save**).

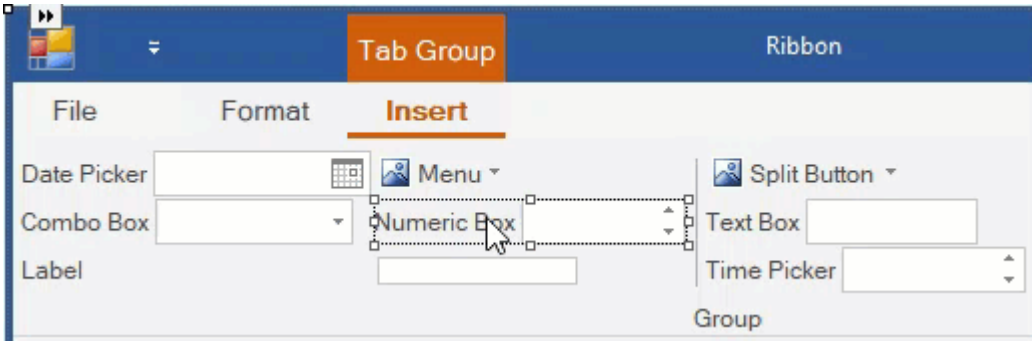


Configuring QAT through Designer

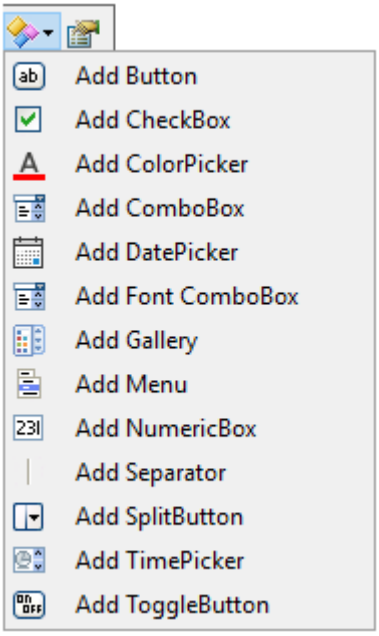
In the designer, the Quick Access Toolbar (QAT) can be customized in just a few clicks. You can configure QAT from the **Items** and **MenuItems** from the **Qat** property in the **Properties** window of **C1Ribbon**.

Further, you can add existing items to the QAT by clicking on the ellipsis, which launches the **QAT Items Collection Editor** or **QAT MenuItems Collection Editor.** To know more about collection editors, refer this topic.

You can add the existing items of the Ribbon control to QAT from the Floating Toolbars of other elements by clicking the QAT icon. The GIF below demonstrates the addition of Numeric box into the QAT using its floating toolbar.



Note that you can also add new elements in the **RibbonQat** using its own floating toolbar. Refer Using Floating Toolbar topic to know more.



Adding Items to QAT via Code

The ribbon buttons can also be added to the QAT programmatically by setting the Items property of the RibbonQat class. This is depicted in the code below:

- **Visual Basic**

```
'アイテムをクイックアクションツールバーに追加します
Dim openImage As Image = Image.FromFile("images\open.png")
customRibbon.Qat.Items.Add(New RibbonButton("Open", openImage))
Dim undoImage As Image = Image.FromFile("images\undo.png")
customRibbon.Qat.Items.Add(New RibbonButton("Undo", undoImage))
Dim redoImage As Image = Image.FromFile("images\redo.png")
customRibbon.Qat.Items.Add(New RibbonButton("Repeat", redoImage))
```

- **C#**

```
//アイテムをクイックアクションツールバーに追加します
Image openImage = Image.FromFile(@"images\open.png");
customRibbon.Qat.Items.Add(new RibbonButton("Open", openImage));
Image undoImage = Image.FromFile(@"images\undo.png");
customRibbon.Qat.Items.Add(new RibbonButton("Undo", undoImage));
Image redoImage = Image.FromFile(@"images\redo.png");
customRibbon.Qat.Items.Add(new RibbonButton("Repeat", redoImage));
Image saveImage = Image.FromFile(@"images\save-file-icon.png");
customRibbon.Qat.Items.Add(new RibbonButton("Save", saveImage));
```

Moving the Quick Access Toolbar

A user can move the QAT between two possible locations, above and below the ribbon control. By default, the QAT is located above the ribbon control. In order to move it down the ribbon, the user can set theBelowRibbon property to True at design time in the Properties Window.

The QAT can also be added below the Ribbon control programmatically using the BelowRibbon property of RibbonQat class as shown below:

- **Visual Basic**

```
'リボンコントロールの下にクイックアクセスツールバーを追加します
customRibbon.Qat.BelowRibbon = True
```

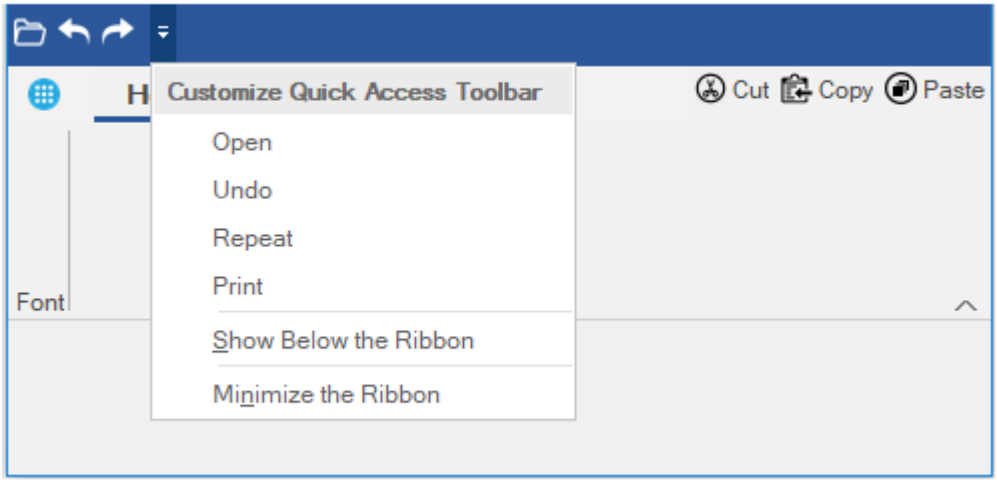
- **C#**

```
//リボンコントロールの下にクイックアクセスツールバーを追加します
customRibbon.Qat.BelowRibbon = true;
```

Customizing QAT Menu

A user can add commands to drop-down menu of the QAT item. This menu will include some commands that the user might want to add to the Quick Access Toolbar.

The image below shows a Ribbon QAT drop-down menu with commands (**Open, Undo, Repeat** and **Print**).



At design time, the user can display the Customize Quick Access Toolbar dropdown menu by setting the **MenuVisible** property in the **Properties** Window to True. Further, the user can click on the **QAT MenuItems Collection Editor** to add buttons on the dropdown menu.

The user can also add the dropdown menu to QAT programmatically using MenuVisible property and MenuItems property of RibbonQat class as shown below:

- **Visual Basic**

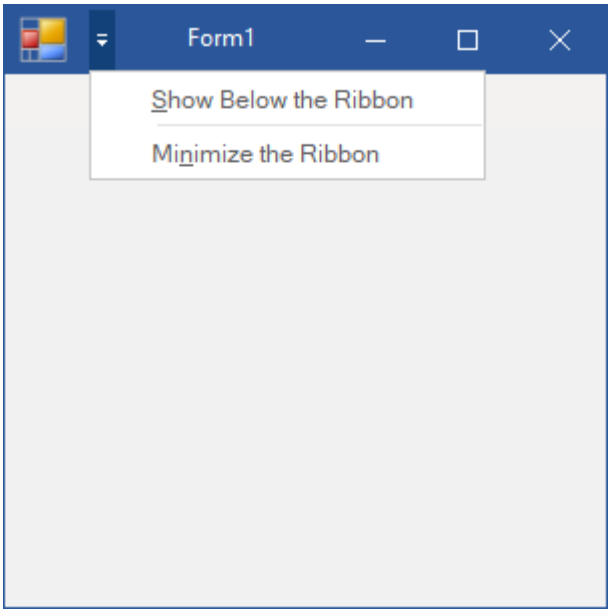
```
'アイテムをQATのメニュードロップダウンに追加します
customRibbon.Qat.MenuItems.Add(New RibbonButton("Open", openImage))
customRibbon.Qat.MenuItems.Add(New RibbonButton("Undo", undoImage))
customRibbon.Qat.MenuItems.Add(New RibbonButton("Repeat", redoImage))
customRibbon.Qat.MenuItems.Add(New RibbonButton("Print", printImage))
```

- **C#**

```
//アイテムをQATのメニュードロップダウンに追加します
customRibbon.Qat.MenuVisible = true;
customRibbon.Qat.MenuItems.Add(new RibbonButton("Open", openImage));
customRibbon.Qat.MenuItems.Add(new RibbonButton("Undo", undoImage));
customRibbon.Qat.MenuItems.Add(new RibbonButton("Repeat", redoImage));
customRibbon.Qat.MenuItems.Add(new RibbonButton("Print", printImage));
```

Hiding Predefined Items from the QAT Toolbar

Ribbon supports hiding the predefined menu items from the Ribbon QAT drop-down. These menu items which appear on clicking the Ribbon drop-down are "**Show Above the Ribbon**" and "**Minimize the Ribbon**" as shown in the following image.



By default, both these options are displayed when you click the QAT drop-down. However, you can choose to hide them. You can set the ShowBelowItem property of the RibbonQat class to false to hide the **Show Below the Ribbon** QAT menu item and ShowMinimizeItem property of the **RibbonQat** class to false to hide the **Minimize the Ribbon** QAT menu item from the Ribbon QAT drop-down as shown in the following code:

Visual Basic

```
C1Ribbon.Qat.ShowBelowItem=False
C1Ribbon.Qat.ShowMinimizeItem=False
```

C#

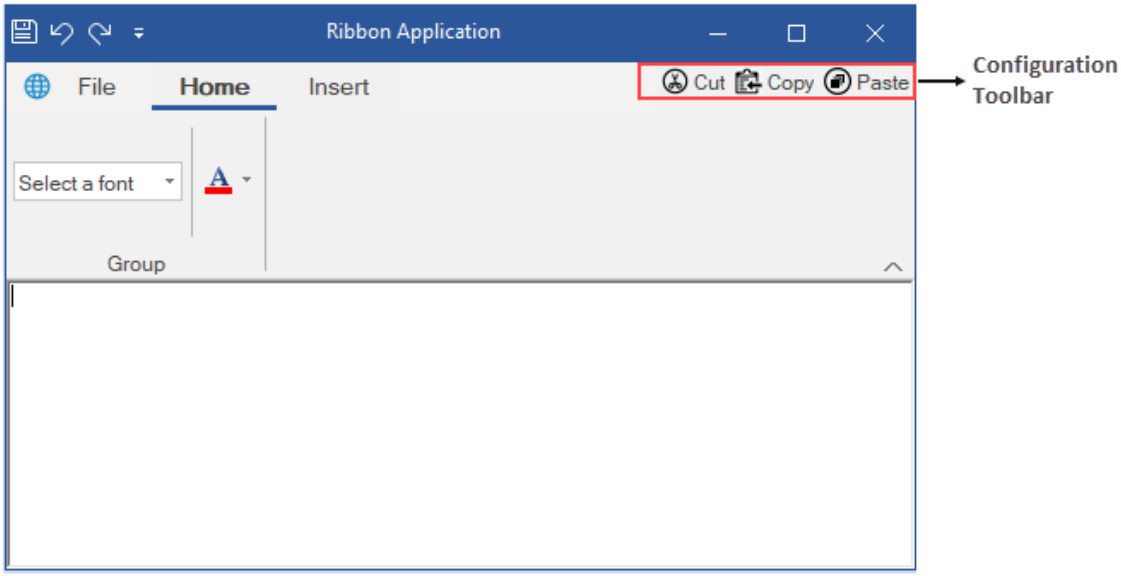
```
C1Ribbon.Qat.ShowBelowItem=false;
C1Ribbon.Qat.ShowMinimizeItem=false;
```

Note: This feature of Ribbon QAT is supported only in .NET 4.5.2.

設定ツールバー

The **Ribbon Configuration Toolbar** (**RibbonConfigToolBar**) allows the user to place commonly-used commands in a toolbar located in the upper-right corner of the Ribbon. Unlike QAT, this toolbar cannot be moved below the Ribbon. The user can observe that it is present at the same level as the Ribbon tabs.

The ribbon control depicted in the image below shows a **Configuration Toolbar** with Ribbon buttons (**Cut**, **Copy** and **Paste**).



Adding Items to Configuration Toolbar

A user can configure Ribbon buttons in the configuration toolbar. For instance, let's say a user wants to perform Cut, Copy or Paste operation in the application. For this, the user has to search through the tabs and groups to find the commands. With the configuration toolbar, you do not have to tediously search for commands. Instead, you can add these commands to the toolbar and customize it.

At design time, you can add items or buttons to the Configuration Toolbar with the help of **Items** property in the Properties Window. The user can click on the **RibbonConfigToolBar Items Collection Editor** to add buttons on the toolbar. Further, the text and icon image of these buttons can be customized using the **Ribbon Button floating toolbar**. For more information, refer this topic.

A user can also add buttons to the Configuration Toolbar programmatically. This is shown in the code below:

- **Visual Basic**

```
Public Sub ConfigToolbarItems(customRibbon As C1Ribbon)
    'デフォルトのConfigToolbarに追加するアイテムを作成します
    Dim cutButton As New RibbonButton("Cut", Image.FromFile("images\cut.png"))
    Dim copyButton As New RibbonButton("Copy", Image.FromFile("images\copy.png"))
    Dim pasteButton As New RibbonButton("Paste", Image.FromFile("images\paste.png"))

    'ConfigToolbarのアイテムを追加します
    customRibbon.ConfigToolBar.Items.Add(cutButton)
    customRibbon.ConfigToolBar.Items.Add(copyButton)
    customRibbon.ConfigToolBar.Items.Add(pasteButton)
End Sub
```

- **C#**

```
public void AddConfigToolbarItems(C1Ribbon customRibbon)
{

    //デフォルトのConfigToolbarに追加するアイテムを作成します
    RibbonButton cutButton = new RibbonButton("Cut", Image.FromFile(@"images\cut.png"));
    RibbonButton copyButton = new RibbonButton("Copy", Image.FromFile(@"images\copy.png"));
    RibbonButton pasteButton = new RibbonButton("Paste", Image.FromFile(@"images\paste.png"));

    //ConfigToolbarのアイテムを追加します
    customRibbon.ConfigToolBar.Items.Add(cutButton);
    customRibbon.ConfigToolBar.Items.Add(copyButton);
    customRibbon.ConfigToolBar.Items.Add(pasteButton);
```

```
}
```

コンテキストタブグループ

The Ribbon control has a **contextual tab group**, which is a hidden tab of groups that appears only when texts or images are selected in your application. Given below is a GIF that depicts the appearance of the contextual tab when a selection of text is being made.

At design-time, a user can customize the look of the contextual tab via the floating tool bar or Collection Editor, about which you can refer in detail in this topic.

This section basically discusses the implementation of contextual tab groups via code in a Ribbon control using the **RibbonContextualTabGroup ('RibbonContextualTabGroup クラス' in the on-line documentation)** class.

- **Visual Basic**

```
'ContextualTabGroupを作成します
Dim selectionSettings As RibbonContextualTabGroup
Private Sub Form1_Load(sender As Object, e As EventArgs)
    'リボンの入力フォーカスを無効にします
    C1Ribbon1.Selectable = False
    'ContextualTabGroupを初期化します
    selectionSettings = New RibbonContextualTabGroup("Format")
    '初期レンダリング時にコンテキストタブグループを非表示にします
    selectionSettings.Visible = False

    'ContextualTabGroupのタブを作成します
    Dim styleSettings As New RibbonTab("Style")
    Dim fontSettings As New RibbonTab("Font")
    selectionSettings.Tabs.Add(styleSettings)
    selectionSettings.Tabs.Add(fontSettings)

    '[スタイル]タブのグループを作成します
    Dim styleGroup As New RibbonGroup("Text Style")
    styleSettings.Groups.Add(styleGroup)

    'RibbonItemsをText Styleグループに追加します
    Dim ribbonToolBar As New RibbonToolBar()
    Dim boldButton As New RibbonToggleButton(Image.FromFile("images\bold.png"))
    Dim italicButton As New RibbonToggleButton(Image.FromFile("images\italic.png"))
    Dim underlineButton As New RibbonToggleButton(Image.FromFile("images\underline.png"))
    ribbonToolBar.Items.Add(boldButton)
    ribbonToolBar.Items.Add(italicButton)
    ribbonToolBar.Items.Add(underlineButton)
    styleGroup.Items.Add(ribbonToolBar)

    '[書式]タブのフォントグループを作成します
    Dim fontGroup As New RibbonGroup("Font Settings")
    fontSettings.Groups.Add(fontGroup)

    'フォントグループにRibbonItemsを作成して追加します
    Dim fontComboBox As New RibbonFontComboBox()
    fontComboBox.Text = "Select a font"
    fontGroup.Items.Add(fontComboBox)

    'C1Ribbonコントロールに ContextualTabGroupを追加します
    C1Ribbon1.ContextualTabGroups.Add(selectionSettings)
End Sub

'テキスト選択に基づいてコンテキストタブを非表示/表示します
```

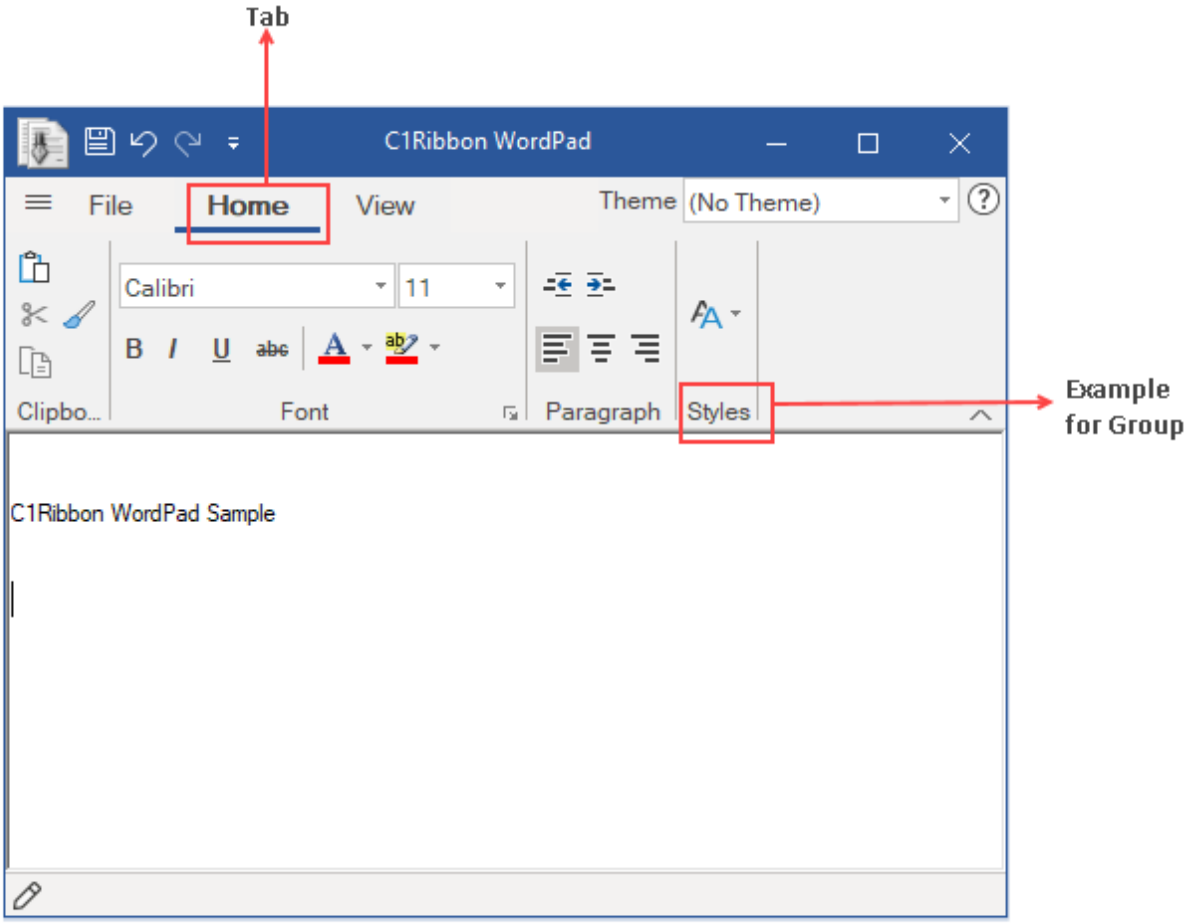
```
Private Sub RichTextBox1_SelectionChanged(ByVal sender As Object, ByVal e As EventArgs)
    If RichTextBox1.SelectedText.Length > 0 Then
        selectionSettings.Visible = True
    Else
        selectionSettings.Visible = False
    End If
End Sub
```

- **C#**

```
RibbonContextualTabGroup selectionSettings;
private void Form1_Load(object sender, EventArgs e)
{
    //リボンの入力フォーカスを無効にします
    c1Ribbon1.Selectable = false;
    //ContextualTabGroupを初期化します
    selectionSettings = new RibbonContextualTabGroup("Format");
    //初期レンダリング時にコンテキストタブグループを非表示にします
    selectionSettings.Visible = false;

    //ContextualTabGroupのタブを作成します
    RibbonTab styleSettings = new RibbonTab("Style");
    RibbonTab fontSettings = new RibbonTab("Font");
    selectionSettings.Tabs.Add(styleSettings);
    selectionSettings.Tabs.Add(fontSettings);

    //[スタイル]タブのグループを作成します
    RibbonGroup styleGroup = new RibbonGroup("Text Style");
    styleSettings.Groups.Add(styleGroup);

    //RibbonItemsをText Styleグループに追加します
    RibbonToolBar ribbonToolBar = new RibbonToolBar();
    RibbonToggleButton boldButton = new RibbonToggleButton(Image.FromFile(@"images\bold.png"));
    RibbonToggleButton italicButton = new RibbonToggleButton(Image.FromFile(@"images\italic.png"));
    RibbonToggleButton underlineButton = new RibbonToggleButton(Image.FromFile(@"images\underline.png"));
    ribbonToolBar.Items.Add(boldButton);
    ribbonToolBar.Items.Add(italicButton);
    ribbonToolBar.Items.Add(underlineButton);
    styleGroup.Items.Add(ribbonToolBar);

    //[書式]タブのフォントグループを作成します
    RibbonGroup fontGroup = new RibbonGroup("Font Settings");
    fontSettings.Groups.Add(fontGroup);

    //フォントグループにRibbonItemsを作成して追加します
    RibbonFontComboBox fontComboBox = new RibbonFontComboBox();
    fontComboBox.Text = "Select a font";
    fontGroup.Items.Add(fontComboBox);

    //C1Ribbonコントロールに ContextualTabGroupを追加します
  c1Ribbon1.ContextualTabGroups.Add(selectionSettings);
}

//テキスト選択に基づいてコンテキストタブを非表示/表示します
private void richTextBox1_SelectionChanged(object sender, EventArgs e)
{
    if (richTextBox1.SelectedText.Length > 0)
        selectionSettings.Visible = true;
    else
        selectionSettings.Visible = false;
}
```

リボンタブ

A user may require many commands in an application in order to work with different scenarios. The commands are usually grouped and put under different **Ribbon Tabs**. A Ribbon tab can contain many groups. Each Group comprises a set of Items. For instance, in the Office application, the Home tab in the Ribbon is used to perform many common operations like changing fonts, applying styles, changing formats, editing texts (copy, cut and paste) etc. The Home tab contains many groups such as Clipboard, Font, Paragraph and Styles.



A tab can be added at design time through the Ribbon control's floating toolbar. A user can add a tab by selecting the **Action** drop down menu of the floating toolbar. The user can add the caption, image and tooltip to the Tab from the floating toolbar. You can refer this topic to understand this in detail. Also, the user can add Ribbon Tabs through the Properties window by accessing the **RibbonTab Collection Editor** in the **Tabs** property. You can refer this topic for more details.

Further, the user can add Tabs in the Ribbon through code with the **RibbonTab ('RibbonTab クラス' in the on-line documentation)** class and **Tabs ('Tabs プロパティ' in the on-line documentation)** property of **C1Ribbon ('C1Ribbon クラス' in the on-line documentation)** class.

- **Visual Basic**

```
'リボンコントロールに新しいタブを作成して追加します
Dim homeTab As New RibbonTab()
'タブにラベルを付けます
homeTab.Text = "Home"
customRibbon.Tabs.Add(homeTab)
```

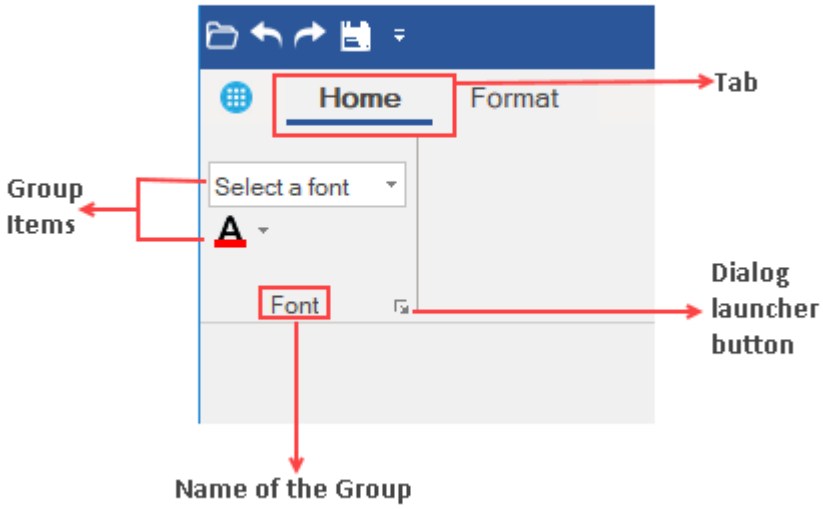
- **C#**

```
//「ホーム」タブを作成して追加します
RibbonTab homeTab = new RibbonTab();
homeTab.Text = "Home";
customRibbon.Tabs.Add(homeTab);
```

リボングループ

All Ribbon Tabs contain Item Groups that can keep many Items together. When a new Tab is added in the C1Ribbon control, it already has a default Group. If the user wants to add more Item Groups, then it can be done by clicking the **Actions** menu of the floating toolbar of the Ribbon Tab. The user can also add a Group with the help of the **RibbonGroup Collection Editor** of Ribbon Tab in Properties window.

The image below depicts a Ribbon Item Group.



You can add Groups to the **C1Ribbon** control using the designer. Refer this topic for more information. The user can also add the Ribbon Item Group programmatically with the **Groups ('Groups プロパティ' in the on-line documentation)** property of **RibbonTab ('RibbonTab クラス' in the on-line documentation)** class and the **Text ('Text プロパティ' in the on-line documentation)** property of **RibbonGroup ('RibbonGroup クラス' in the on-line documentation)** class. This is shown in the code below:

- **Visual Basic**

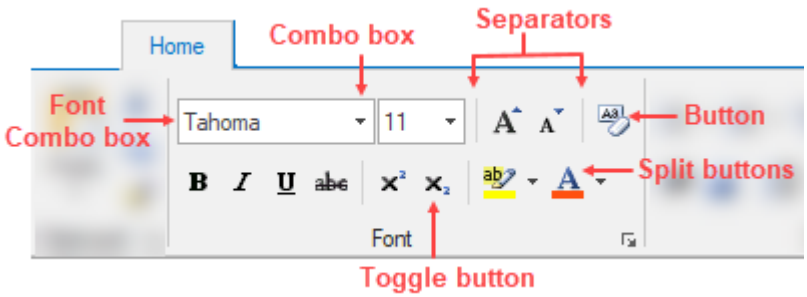
```
'[ホーム]タブにグループを追加します
Dim fontStyle As New RibbonGroup()
'グループにラベルを付けます
fontStyle.Text = "Font"
homeTab.Groups.Add(fontStyle)
```

- **C#**

```
//「フォント」という名前のグループを「ホーム」タブに追加します
RibbonGroup fontStyle = new RibbonGroup();
fontStyle.Text = "Font";
homeTab.Groups.Add(fontStyle);
fontStyle.HasLauncherButton = true;
```

リボンアイテム

A Ribbon group comprises group items which perform a specified command or action. There are various types of group items available in the Ribbon group, such as buttons, check boxes, combo boxes, toolbars, menus, and so on.



The next section will take you through different RibbonGroup items.

Button

CheckBox

Color Picker

ComboBox

Date Picker

Font ComboBox

Gallery

Label

Menu

Numeric Box

Progress Bar

Separator

Split Button

TextBox

Time Picker

Toggle Button

Tool Bar

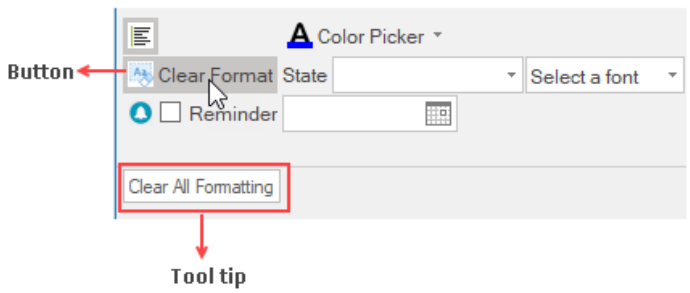
Track Bar

Control Host

ボタン

A Button is a clickable ribbon item that executes a command. Inside a button item, you can put any text or image as required.

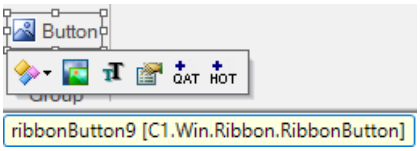
The image below displays a ribbon application with **Clear Format** button and tooltip.



Add Ribbon Button at Design-Time

The **Ribbon Button** can be added at design-time using the **Ribbon Group Floating Toolbar** or **RibbonGroup Items Collection Editor**. Also, you can customize the look of the Ribbon Button using the **Ribbon Button Floating ToolBar** or by editing the properties in the Properties Window. For more info on floating toolbars, refer this topic.

This image below shows the floating toolbar of Button.



Add Button via Code

A ribbon button can also be added to the **C1Ribbon** control through the code. This can be done by using the RibbonButton class.

- **Visual Basic**

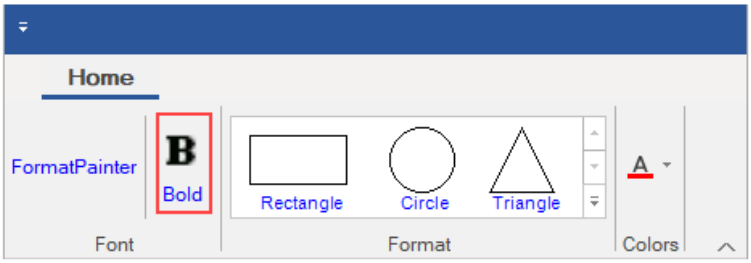
```
' Add ribbon button to the format group
Dim clearButton As RibbonButton = New RibbonButton("Clear Format", Image.FromFile("images\clearformat.png"))
clearButton.ToolTip = "Clear All Formatting"
formatGroup.Items.Add(clearButton)
```

- **C#**

```
// Add ribbon button to the format group
RibbonButton clearButton = new RibbonButton("Clear Format", Image.FromFile(@"images\clearformat.png"));
clearButton.ToolTip = "Clear All Formatting";
formatGroup.Items.Add(clearButton);
```

Change ForeColor

The ForeColor of an item typically refers to its text color. Users can change the ForeColor of the ribbon button item. The **C1Ribbon** class provides the **UpdatingItemStyle** event, which occurs before a style is applied to a ribbon item.



Let's see how to change the ForeColor of RibbonButton by setting the UpdatingItemStyle event in the code snippet.

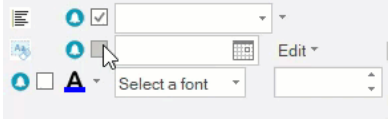
C# copyCode

```
//Handle UpdatingItemStyle event to set forecolor of RibbonButton
private void C1Ribbon1_UpdatingItemStyle(object sender, UpdatingItemStyleEventArgs e)
{
    if (e.RibbonItem.GetType() == typeof(RibbonButton))
    {
        e.ForeColor = Color.Blue;
    }
}
```

チェックボックス

CheckBoxes are helpful when there are multiple options appearing in a list. It can be used to turn an option on or off.

The following ribbon group contains three CheckBox items.



The CheckBox can be added at design-time using the Floating Toolbar or Collection Editor. Refer this topic for detailed information.

The CheckBox can also be added to the **C1Ribbon** control programmatically. This can be done using the **RibbonCheckBox ('RibbonCheckBox クラス' in the on-line**

documentation) class.

- **Visual Basic**

```
' リボン項目のCheckBoxボタンを追加します
Dim reminderCheckBox As RibbonCheckBox = New RibbonCheckBox()
reminderCheckBox.IconSet.Add(New C1BitmapIcon(Nothing, New Size(20, 20), Color.Transparent, Image.FromFile("images\reminder.png")))
reminderCheckBox.Text = "Reminder"
formatGroup.Items.Add(reminderCheckBox)
```

- **C#**

```
// リボン項目のCheckBoxボタンを追加します
RibbonCheckBox reminderCheckBox1 = new RibbonCheckBox();
reminderCheckBox1.IconSet.Add(new C1BitmapIcon(null, new Size(20, 20), Color.Transparent, Image.FromFile(@"images\reminder.png")));
reminderCheckBox1.Text ="Thumbnails";
formatGroup.Items.Add(reminderCheckBox1);

RibbonCheckBox reminderCheckBox2 = new RibbonCheckBox();
reminderCheckBox2.IconSet.Add(new C1BitmapIcon(null, new Size(20, 20), Color.Transparent, Image.FromFile(@"images\reminder.png")));
reminderCheckBox2.Text = "Document map";
formatGroup.Items.Add(reminderCheckBox2);
```

カラーピッカー

A color picker is a button which when clicked displays a drop-down color palette. The user can select a specific color from the preset palette by clicking the color picker's drop-down arrow.

The **Color Picker** can be added at design-time using the floating toolbar or Collection Editor. Refer this topic for detailed information.

A ColorPicker can also be added to the C1Ribbon control through the code using the **RibbonColorPicker ('RibbonColorPicker クラス' in the on-line documentation)** class. This is depicted in the code below:

- **Visual Basic**

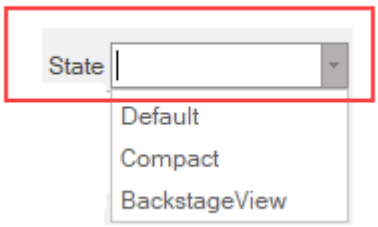
```
'リボン項目のColorPickerを追加します
Dim colorPicker As RibbonColorPicker = New RibbonColorPicker("Color Picker", Image.FromFile("images\fontcolor.png"))
colorPicker.DefaultColor = Color.Blue
formatGroup.Items.Add(colorPicker)
```

- **C#**

```
// リボン項目のColorPickerを追加します
RibbonColorPicker colorPicker = new RibbonColorPicker("Color Picker", Image.FromFile(@"images\fontcolor.png"));
colorPicker.DefaultColor = Color.Blue;
formatGroup.Items.Add(colorPicker);
```

コンボボックス

A ComboBox is a combination of a single-line text box with drop-down list. The **ComboBox** item in this is highlighted below:



The **ComboBox** can be added at design-time using the Floating Toolbar or Collection Editor. Refer this topic for detailed information.

A ComboBox can also be added to the **C1Ribbon** control through the code using the **RibbonComboBox ('RibbonComboBox クラス' in the on-line documentation)** class. This is depicted in the code below:

- **Visual Basic**

```
' リボン項目のComboBoxを追加します
Dim ribbonComboBox As RibbonComboBox = New RibbonComboBox()
ribbonComboBox.Label = "State"
```

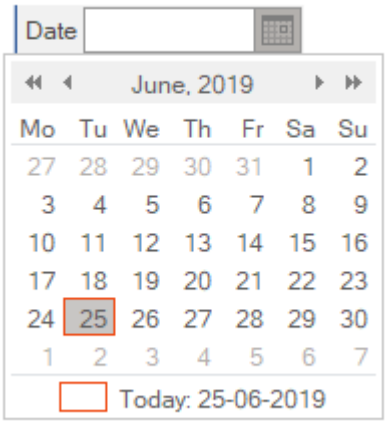
```
ribbonComboBox.TextAreaWidth = 100
Dim defaultView As RibbonButton = New RibbonButton("Default")
Dim comapactView As RibbonButton = New RibbonButton("Compact")
Dim backstageView As RibbonButton = New RibbonButton("BackstageView")
ribbonComboBox.Items.Add(defaultView)
ribbonComboBox.Items.Add(comapactView)
ribbonComboBox.Items.Add(backstageView)
formatGroup.Items.Add(ribbonComboBox)
```

- **C#**

```
//リボン項目のComboBoxを追加します
RibbonComboBox ribbonComboBox = new RibbonComboBox();
ribbonComboBox.Label = "State";
ribbonComboBox.TextAreaWidth = 100;
RibbonButton defaultView = new RibbonButton("Default");
RibbonButton comapactView = new RibbonButton("Compact");
RibbonButton backstageView = new RibbonButton("BackstageView");
ribbonComboBox.Items.Add(defaultView);
ribbonComboBox.Items.Add(comapactView);
ribbonComboBox.Items.Add(backstageView);
formatGroup.Items.Add(ribbonComboBox);
```

日付ピッカー

A date picker allows a user to choose a specific date from a drop-down calendar or enter a specific date in the numeric box. The user can click the DatePicker's calendar icon to select a date.



The **Date Picker** can be added at design-time using the Floating Toolbar or Collection Editor. Refer this topic for detailed information.

The date picker item can also be added to the **C1Ribbon** control through the code using the **RibbonDatePicker ('RibbonDatePicker クラス' in the on-line documentation)** class. This is depicted in the code below:

- **Visual Basic**

```
' リボン項目のDatePickerを追加します
Dim datePicker As RibbonDatePicker = New RibbonDatePicker()
datePicker.Format = "yyyy/MM/dd"
formatGroup.Items.Add(datePicker)
```

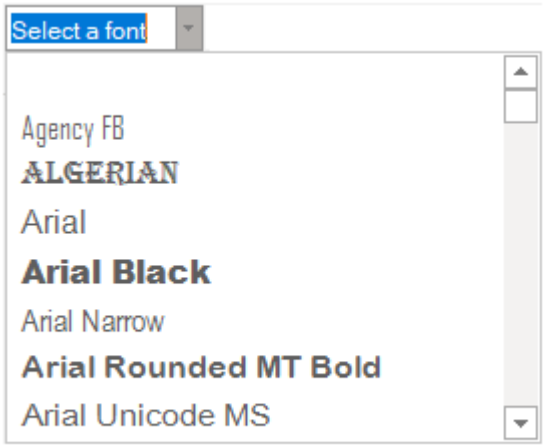
- **C#**

```
// リボン項目のDatePickerを追加します
RibbonDatePicker datePicker = new RibbonDatePicker();
```

```
datePicker.Format = "yyyy/MM/dd";
formatGroup.Items.Add(datePicker);
```

フォントコンボボックス

The Font ComboBox is a combination of drop-down list and list box that shows the available font styles. It also provides a single-line text box. The user can click the Font ComboBox to view the dropdown list of available font styles.



The **Font ComboBox** can be added at design-time using the Floating Toolbar or Collection Editor. Refer this topic for detailed information.

A Font ComboBox can also be added to the C1Ribbon control through the code using the **RibbonFontComboBox ('RibbonFontComboBox クラス' in the on-line documentation)** class. This is depicted in the code below:

- **Visual Basic**

```
'リボン項目のFontComboBox を追加します
Dim fontComboBox As RibbonFontComboBox = New RibbonFontComboBox()
fontComboBox.Text = "Select a font"
fontComboBox.AutoCompleteMode = ComboBoxAutoCompleteMode.Suggest
formatGroup.Items.Add(fontComboBox)
```

- **C#**

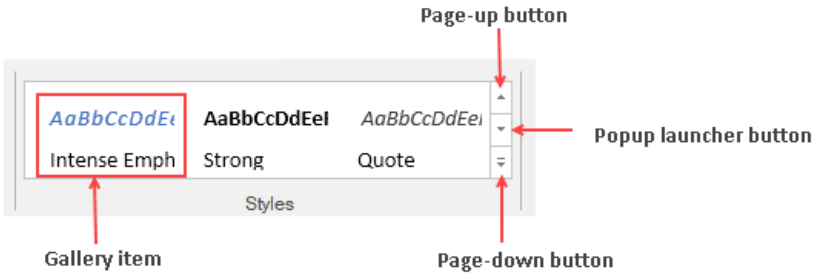
```
// リボン項目のFontComboBox を追加します
RibbonFontComboBox fontComboBox = new RibbonFontComboBox();
fontComboBox.Text = "Select a font";
fontComboBox.AutoCompleteMode = ComboBoxAutoCompleteMode.Suggest;
formatGroup.Items.Add(fontComboBox);
```

ギャラリー

Gallery displays a collection of related commands in a Ribbon. The **Gallery** contains a list of clickable items. Each item when clicked shows the result visually on the selected text.

The GIF below illustrates how the text changes upon applying various Styles from the Gallery.

The image below shows a gallery item in the **C1Ribbon** control.

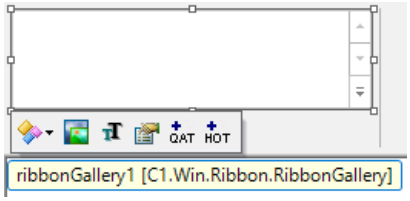


The **Ribbon Gallery** has a page-up and page-down button to scroll up and down the gallery. There is also a popup launcher button that opens a window with ribbon items.

Configuring Gallery at Design-Time

The **Ribbon Gallery** can be added at design-time using the **Ribbon Group Floating Toolbar** or **RibbonGroup Items Collection Editor**. Also, you can customize the look of the Ribbon Gallery using the **Ribbon Gallery Floating ToolBar** or by editing the properties of Gallery in the Properties Window. Refer this topic, for more information on floating toolbars. You can add items to Gallery using the **RibbonGallery Items Collection Editor** and **RibbonGallery MenuItems Collection Editor**. For more info on Collection Editors, refer this topic.

The image below shows the floating toolbar of Gallery.



Adding Ribbon Gallery and Ribbon Gallery Items

The user can also add **Gallery** and **Gallery items** in the C1Ribbon control through code using the **RibbonGallery ('RibbonGallery クラス' in the on-line documentation)** class and **RibbonGalleryItem ('RibbonGalleryItem クラス' in the on-line documentation)** class. For instance, the code below depicts how to add a gallery of shapes to the Ribbon control.

- **Visual Basic**

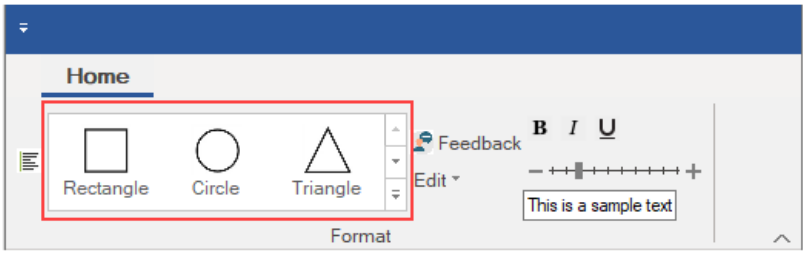
```
' リボン項目のRibbonGalleryを追加します
Dim shapesGallery As RibbonGallery = New RibbonGallery()
shapesGallery.ToolTip = "Select a shape"
Dim rectShape As RibbonGalleryItem = New RibbonGalleryItem("Rectangle", Image.FromFile("images\rect.png"))
Dim circleShape As RibbonGalleryItem = New RibbonGalleryItem("Circle", Image.FromFile("images\circle.png"))
Dim triangleShape As RibbonGalleryItem = New RibbonGalleryItem("Triangle", Image.FromFile("images\triangle.png"))
Dim hexagonShape As RibbonGalleryItem = New RibbonGalleryItem("Hexagon", Image.FromFile("images\hexagon.png"))
shapesGallery.Items.Add(rectShape)
shapesGallery.Items.Add(circleShape)
shapesGallery.Items.Add(triangleShape)
shapesGallery.Items.Add(hexagonShape)
```

```
formatGroup.Items.Add(shapesGallery)
```

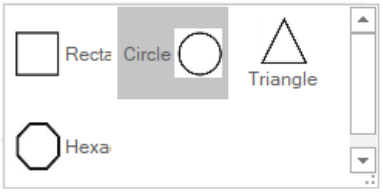
- **C#**

```
// リボン項目のRibbonGalleryを追加します
RibbonGallery shapesGallery = new RibbonGallery();
shapesGallery.ToolTip = "Select a shape";
RibbonGalleryItem rectShape = new RibbonGalleryItem("Rectangle", Image.FromFile(@"images\rect.png"));
RibbonGalleryItem circleShape = new RibbonGalleryItem("Circle", Image.FromFile(@"images\circle.png"));
RibbonGalleryItem triangleShape = new RibbonGalleryItem("Triangle", Image.FromFile(@"images\triangle.png"));
RibbonGalleryItem hexagonShape = new RibbonGalleryItem("Hexagon", Image.FromFile(@"images\hexagon.png"));
shapesGallery.Items.Add(rectShape);
shapesGallery.Items.Add(circleShape);
shapesGallery.Items.Add(triangleShape);
shapesGallery.Items.Add(hexagonShape);
formatGroup.Items.Add(shapesGallery);
```

The snapshot of the resulting ribbon control is shown below:



The user can change the position of the image and text of each gallery item. You can place the image above, before or after the text using the **ImageAboveText**, **ImageBeforeText and ImageAfterText** properties of **GalleryItemTextImageRelation** enum.



You can also assign the position of image and text of each item in the gallery via code:

- **Visual Basic**

```
' Specify the position of text And image for gallery items
ribbonGallery1.GalleryItemTextImageRelation = GalleryItemTextImageRelation.ImageBeforeText
ribbonGallery2.GalleryItemTextImageRelation = GalleryItemTextImageRelation.ImageAfterText
ribbonGallery3.GalleryItemTextImageRelation = GalleryItemTextImageRelation.ImageAboveText
ribbonGallery4.GalleryItemTextImageRelation = GalleryItemTextImageRelation.ImageBeforeText
```

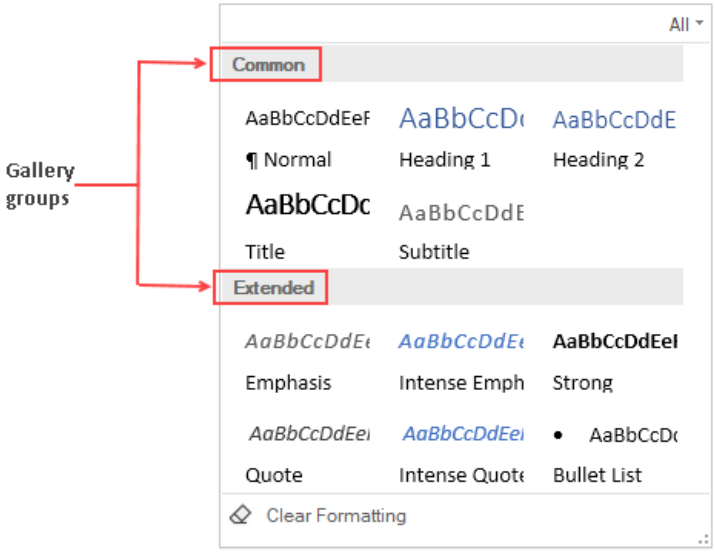
- **C#**

```
// Specify the position of text and image for gallery items
ribbonGallery1.GalleryItemTextImageRelation = GalleryItemTextImageRelation.ImageBeforeText;
ribbonGallery2.GalleryItemTextImageRelation = GalleryItemTextImageRelation.ImageAfterText;
ribbonGallery3.GalleryItemTextImageRelation = GalleryItemTextImageRelation.ImageAboveText;
ribbonGallery4.GalleryItemTextImageRelation = GalleryItemTextImageRelation.ImageBeforeText;
```

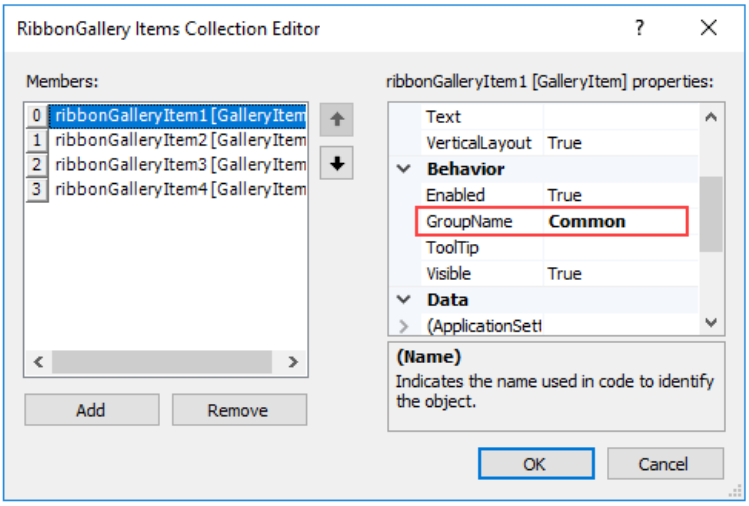
Gallery Grouping

If you click on the popup launcher button, it displays a small window of gallery items, which are grouped by group names.

The image below shows the popup gallery window with items arranged in two groups, **Common** and **Extended**.



Grouping in gallery can be done at design time by assigning **GroupName** to each gallery item in the **RibbonGallery Items Collection Editor**.



You can also assign group names to gallery items programmatically using the **GroupName ('GroupName プロパティ' in the on-line documentation)** property of **RibbonGalleryItem ('RibbonGalleryItem クラス' in the on-line documentation)** class. This is shown in the code snippet below:

- **Visual Basic**

```
'グループにギャラリー項目を追加します
galleryitem1.GroupName = "Common"
galleryitem2.GroupName = "Common"
galleryitem3.GroupName = "Extended"
galleryitem4.GroupName = "Extended"
```

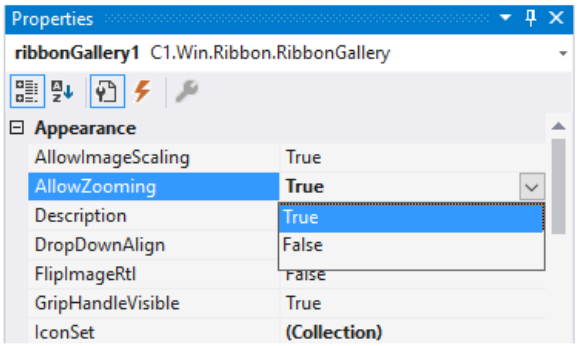
- **C#**

```
//グループにギャラリー項目を追加します
galleryitem1.GroupName = "Common";
galleryitem2.GroupName = "Common";
galleryitem3.GroupName = "Extended";
galleryitem4.GroupName = "Extended";
```

Gallery Zooming

The **C1Ribbon** control supports zooming within the Gallery. The user can rest the cursor on each item and get a zoomed preview of that item image.

At design time, the zooming feature can be activated using the **AllowZooming** property of **Ribbon Gallery** in the **Properties** window. Note that for the zooming property to work, the **ItemSize** property of the Gallery should be less than the image size of each item.



You can also add the zooming feature via code using the **AllowZooming ('AllowZooming プロパティ' in the on-line documentation)** property of **RibbonGallery ('RibbonGallery クラス' in the on-line documentation)** class.

- **Visual Basic**

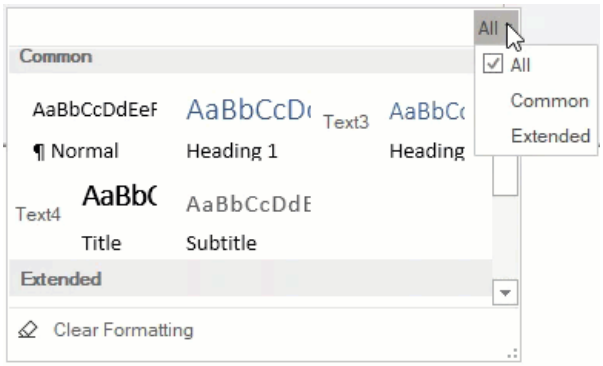
```
'ギャラリーのズームを許可します
ribbonGallery.AllowZooming = True
```

- **C#**

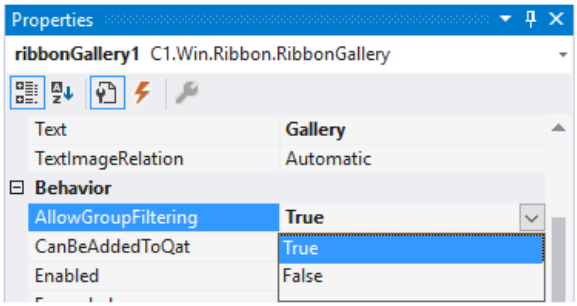
```
//ギャラリーのズームを許可します
ribbonGallery.AllowZooming = true;
```

Gallery Filtering

The **C1Ribbon** control supports filtering in the ribbon gallery. It allows you to filter items by gallery groups. This is illustrated in the GIF below:



At design time, the filtering feature can be activated using the **AllowGroupFiltering** property of **Ribbon Gallery** in the **Properties** window.



You can also add the filtering feature via code using the **AllowGroupFiltering ('AllowGroupFiltering プロパティ' in the on-line documentation)** property of **RibbonGallery ('RibbonGallery クラス' in the on-line documentation)** class.

- **Visual Basic**

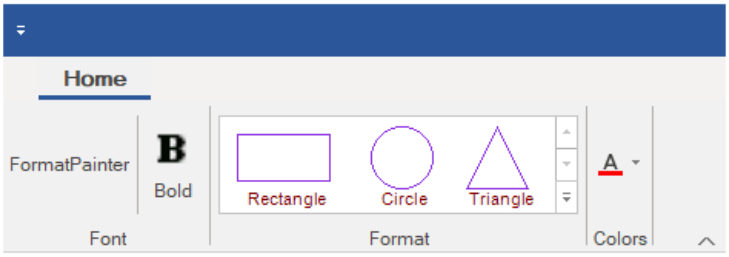
```
'ギャラリーグループでのフィルタリングを許可します
ribbonGallery.AllowGroupFiltering = True
```

- **C#**

```
//ギャラリーグループでのフィルタリングを許可します
ribbonGallery.AllowGroupFiltering = true;
```

Owner-Drawn Gallery

With 'Owner-Drawn' feature, the items are not drawn with the default logic of Ribbon, rather they are drawn with a custom logic defined by the user of the application. This is useful in cases where complex customizations need to be performed. You can render the contents of the RibbonGallery as Owner Drawn by writing a handler for the **DrawItem** event. This event is raised as needed to display the contents of Gallery when its **OwnerDraw** property is set to True.



To create the owner-drawn RibbonGallery in the above illustration, follow the steps below:

1. Add the following code the **Form_Load** event:

C# copyCode

```
//Set OwnerDraw property and bind DrawItem event
//to owner draw RibbonGallery items
ribbonGallery.OwnerDraw = true;
ribbonGallery.DrawItem += RibbonGallery_DrawItem;
```

2. Implement the **DrawItem** event:

C# copyCode

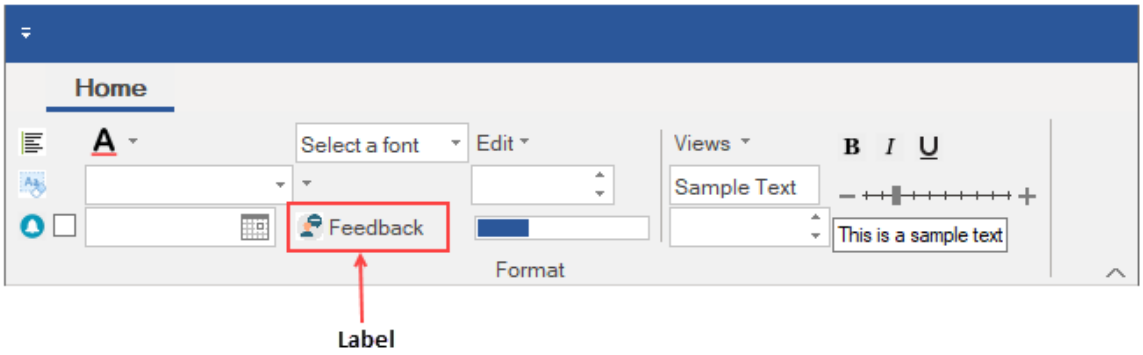
```
//Handle DrawItem event to owner draw RibbonGallery Items
private void RibbonGallery_DrawItem(object sender, C1.Win.Ribbon.DrawItemEventArgs e)
{
    switch (e.Item.Name)
    {
        case "Rectangle":
            Rectangle srect = new Rectangle(e.Bounds.X + 10, e.Bounds.Y + 10, 60, 30);
            e.Graphics.DrawRectangle(new Pen(Color.BlueViolet), srect);
            e.Graphics.DrawString("Rectangle", new Font("Arial", 8), new SolidBrush(Color.Maroon), new
PointF(e.Bounds.X + 15, e.Bounds.Y + 45));
            break;

        case "Circle":
            Rectangle crect = new Rectangle(e.Bounds.X + 25, e.Bounds.Y + 5, 40, 40);
            e.Graphics.DrawEllipse(new Pen(Color.BlueViolet), crect);
            e.Graphics.DrawString("Circle", new Font("Arial", 8), new SolidBrush(Color.Maroon), new
PointF(e.Bounds.X + 30, e.Bounds.Y + 45));
            break;
```

```
        case "Triangle":
            Point point1 = new Point(e.Bounds.X + 35, e.Bounds.Y + 5);
            Point point2 = new Point(e.Bounds.X + 15, e.Bounds.Y + 45);
            Point point3 = new Point(e.Bounds.X + 55, e.Bounds.Y + 45);
            Point[] curvePoints = { point1, point2, point3 };
            e.Graphics.DrawPolygon(new Pen(Color.BlueViolet), curvePoints);
e.Graphics.DrawString("Triangle", new Font("Arial", 8), new SolidBrush(Color.Maroon), new
PointF(e.Bounds.X + 15, e.Bounds.Y + 45));
            break;
    }
}
```

ラベル

A label displays content and can point to the element next to it. It can include both text and image.

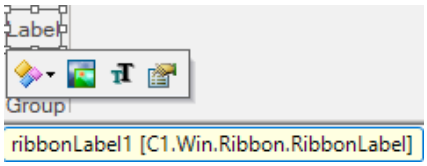
The image given below depicts a label.



Add Label at Design-Time

The **Ribbon Label** can be added at design-time using the **Ribbon Group Floating Toolbar** or **RibbonGroup Items Collection Editor**. Also, you can customize the look of the Ribbon Label using the **Ribbon Label Floating ToolBar** or by editing the properties in the **Properties** Window. For more information about floating toolbars, refer this topic.

This image below shows the floating toolbar of Label.



Add Label via Code

The Label can also be added to the C1Ribbon control through the code using the RibbonLabel class. This is depicted in the code below:

- **Visual Basic**

```
' Add Label ribbon item
Dim ribbonLabel As RibbonLabel = New RibbonLabel("Feedback", Image.FromFile("images\feedback.png"))
formatGroup.Items.Add(ribbonLabel)
```

- **C#**

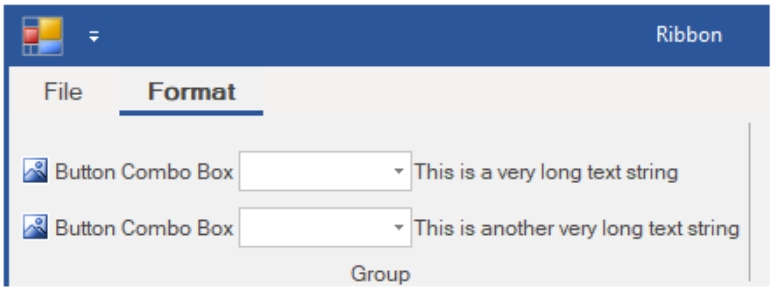
```
// Add a Label ribbon item
RibbonLabel ribbonLabel = new RibbonLabel("Feedback", Image.FromFile(@"images\feedback.png"));
formatGroup.Items.Add(ribbonLabel);
```

Add Multiple Labels

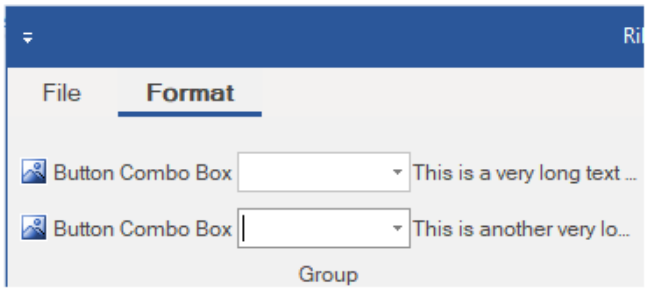
C1Ribbon allows you to align multiple labels within your **RibbonGroup** using the MaxTextWidth and MinTextWidth properties. This

section will walk you through the steps of adding multiple labels using two RibbonToolBars containing the default Ribbonlabel, a RibbonComboBox, and a RibbonLabel.

1. Add two **RibbonToolBars** to your application. By default, the **RibbonToolBar** contains a **Ribbonlabel**.

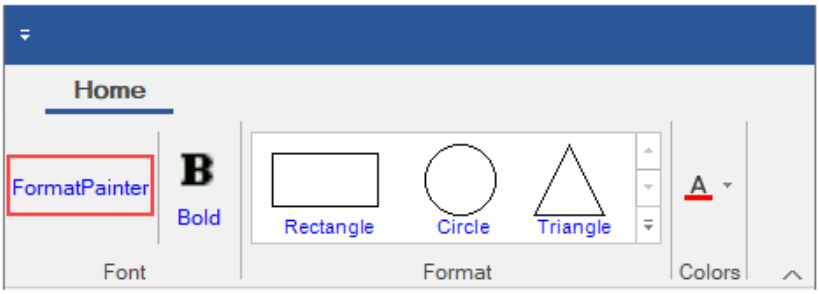


2. Float over the upper left corner of the first Toolbar until the glyph appears. Click the **Action** label and select **Add ComboBox** from the list.
3. Click the glyph again and click the **Action** label again. Select **Add Label** from the list.
4. Repeat steps 1- 3 with the second Toolbar.
5. The Ribbon application should appear as in the following image:
6. Select the **RibbonLabel** on the first Toolbar and choose **Text Settings** from the RibbonLabel Toolbar. In the Text textbox, enter **This is a very long text string**.
7. In the **RibbonLabel** Properties window, scroll down to the **MaxTextWidth** property and set it to 130. Enter the same number for the **MinTextWidth** property.
8. Select the **RibbonLabel** on the second Toolbar and choose Text Settings from the RibbonLabel Toolbar. In the Text textbox, enter **This is another very long text string**.
9. In the **RibbonLabel** Properties window, scroll down to the **MaxTextWidth** property and set it to 130. Enter the same number for the MinTextWidth property.
10. Press **F5** to run your application. Your application should resemble the following image:



Change ForeColor

The ForeColor of an item typically refers to its text color. Users can change the ForeColor of the ribbon label item. The **C1Ribbon** class provides the **UpdatingItemStyle** event, which occurs before a style is applied to a ribbon item.



Let's see how to change the ForeColor of RibbonLabel by setting the UpdatingItemStyle event in the code snippet.

C# copyCode

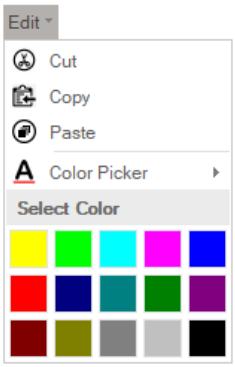
```
//Handle UpdatingItemStyle event to set forecolor of RibbonLabel
```

```
private void C1Ribbon1_UpdatingItemStyle(object sender, UpdatingItemStyleEventArgs e)
{
    if (e.RibbonItem.GetType() == typeof(RibbonLabel))
    {
        e.ForeColor = Color.Blue;
    }
}
```

メニュー

A Menu is a button with a drop-down arrow. It is used in a ribbon application when you need a menu for a small set of related commands. If a user clicks on the arrow, then a dropdown of items is displayed.

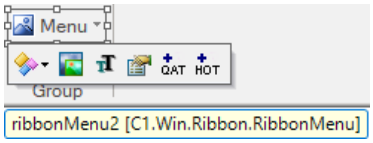
The image below depicts the **Menu** item with commands such as Cut, Copy, Paste and ColorPicker.



Adding Menu at Design-Time

The **Ribbon Menu** can be added at design-time using the **Ribbon Group Floating Toolbar** or **RibbonGroup Items Collection Editor**. Also, you can customize the look of the Ribbon Menu using the **Ribbon Menu Floating ToolBar** or by editing the properties in the **Properties** Window. For more information about floating toolbars, refer this topic. The user can add items to the Menu using the **Ribbon Menu Items Collection Editor**. Refer this topic for more information on collection editors.

This image below shows the floating toolbar of Menu.



Adding Menu via Code

A Menu can also be added to the **C1Ribbon** control through the code using the **RibbonMenu ('RibbonMenu クラス' in the on-line documentation)** class. This is depicted in the code below:

- **Visual Basic**

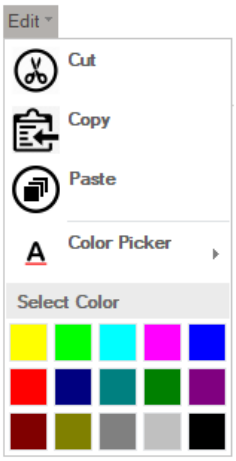
```
' リボン項目のRibbonMenuを追加します
Dim ribbonMenu As RibbonMenu = New RibbonMenu()
ribbonMenu.Text = "Edit"
Dim cutButton As RibbonButton = New RibbonButton("Cut", Image.FromFile("images\cut.png"))
Dim copyButton As RibbonButton = New RibbonButton("Copy", Image.FromFile("images\copy.png"))
Dim pasteButton As RibbonButton = New RibbonButton("Paste", Image.FromFile("images\paste.png"))
Dim separator As RibbonSeparator = New RibbonSeparator()
Dim colorpicker1 As RibbonColorPicker = New RibbonColorPicker("Color Picker", Image.FromFile("images\fontcolor.png"))
Dim label As RibbonLabel = New RibbonLabel("Select Color")
Dim colorItem As RibbonColorPickerItem = New RibbonColorPickerItem()
ribbonMenu.Items.Add(cutButton)
ribbonMenu.Items.Add(copyButton)
ribbonMenu.Items.Add(pasteButton)
ribbonMenu.Items.Add(separator)
ribbonMenu.Items.Add(colorPicker)
ribbonMenu.Items.Add(label)
ribbonMenu.Items.Add(colorItem)
formatGroup.Items.Add(ribbonMenu)
```

- **C#**

```
// リボン項目のRibbonMenuを追加します
RibbonMenu ribbonMenu = new RibbonMenu();
ribbonMenu.Text = "Edit";
```

```
RibbonButton cutButton = new RibbonButton("Cut", Image.FromFile(@"images\cut.png"));
RibbonButton copyButton = new RibbonButton("Copy", Image.FromFile(@"images\copy.png"));
RibbonButton pasteButton = new RibbonButton("Paste",Image.FromFile(@"images\paste.png"));
RibbonSeparator separator = new RibbonSeparator();
RibbonColorPicker colorpicker = new RibbonColorPicker("Color Picker", Image.FromFile(@"images\fontcolor.png"));
RibbonLabel label = new RibbonLabel("Select Color");
RibbonColorPickerItem colorItem = new RibbonColorPickerItem();
ribbonMenu.Items.Add(cutButton);
ribbonMenu.Items.Add(copyButton);
ribbonMenu.Items.Add(pasteButton);
ribbonMenu.Items.Add(separator);
ribbonMenu.Items.Add(colorpicker);
ribbonMenu.Items.Add(label);
ribbonMenu.Items.Add(colorItem);
formatGroup.Items.Add(ribbonMenu);
```

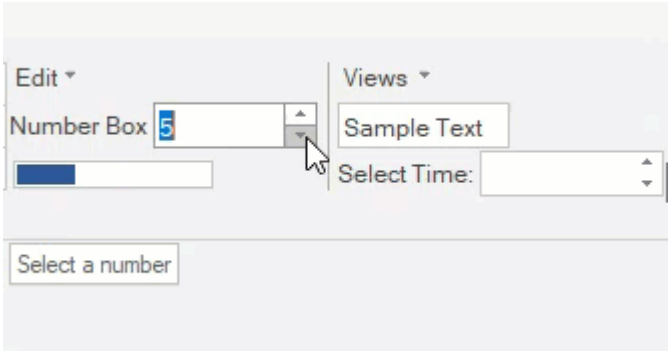
A user can specify the item size in the dropdown menu by specifying the **PreferredItemSize** property. The image below depicts the Menu item in the ribbon control when the **PreferredItemSize** property is set to Large.



数値ボックス

A NumericBox displays a box in which numeric values can be entered. The limits for **Numeric Box** can be set using the **Minimum ('Minimum プロパティ' in the on-line documentation)** and **Maximum ('Maximum プロパティ' in the on-line documentation)** properties.

The GIF below depicts a NumericBox with minimum and maximum values of 5 and10 respectively.



The Numeric Box can be added at design-time using the Floating Toolbar or Collection Editor. Refer this topic for detailed information. A NumericBox can also be added to the **C1Ribbon** control through the code using the **RibbonNumericBox ('RibbonNumericBox クラス' in the on-line documentation)** class. This is depicted in the code below:

- **Visual Basic**

```
' リボン項目の数値ボックスを追加します
Dim numericBox As RibbonNumericBox = New RibbonNumericBox()
```

```
numericBox.Maximum = 10
numericBox.Minimum = 5
numericBox.ToolTip = "Select a number"
numericBox.Label = "Number Box"
formatGroup.Items.Add(numericBox)
```

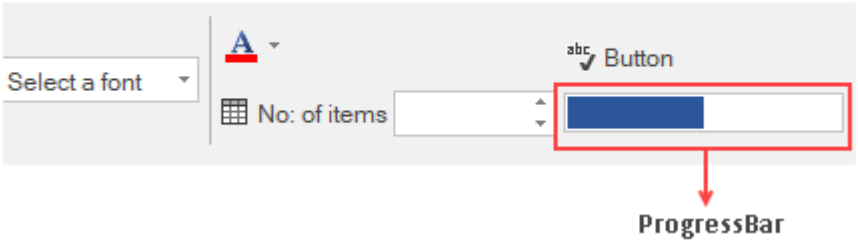
- **C#**

```
// リボン項目の数値ボックスを追加します
RibbonNumericBox numericBox = new RibbonNumericBox();
numericBox.Maximum = 10;
numericBox.Minimum = 5;
numericBox.ToolTip = "Select a number";
numericBox.Label = "Number Box";
formatGroup.Items.Add(numericBox);
```

プログレスバー

A Progress Bar is a **C1Ribbon** Item used to visualize the progression of an extended task occurring within an application. The user can assign **Minimum ('Minimum プロパティ' in the on-line documentation)**, **Maximum ('Maximum プロパティ' in the on-line documentation)** and **Value ('Value プロパティ' in the on-line documentation)** properties to the ProgressBar range.

The image below depicts a Progress Bar with a Minimum and Maximum of 0 and 100 respectively. The current Value is shown as 50 here, indicating that the task is partly completed.



The ProgresBar can be added at design-time using the Floating Toolbar or Collection Editor. Refer this topic for detailed information. A ProgressBar can also be added to the C1Ribbon control through the code using the **RibbonProgressBar ('RibbonProgressBar クラス' in the on-line documentation)** class. This is depicted in the code below:

- **Visual Basic**

```
' リボン項目のProgressBarを追加します
Dim progressBar As RibbonProgressBar = New RibbonProgressBar()
progressBar.Minimum = 0
progressBar.Maximum = 100
progressBar.Value = 50
formatGroup.Items.Add(progressBar)
```

- **C#**

```
// リボン項目のプログレスバーを追加します
RibbonProgressBar progressBar = new RibbonProgressBar();
progressBar.Minimum = 0;
progressBar.Maximum = 100;
progressBar.Value = 50;
progressBar.ToolTip = "50%";
formatGroup.Items.Add(progressBar);
```

区切り記号

The separator item is a horizontal or vertical bar used to provide visual separation between items/commands in a group, toolbar, StatusBar or drop-down menus.

The separator can be added at design-time using the Floating Toolbar or Collection Editor. Refer this topic for detailed information. A separator can also be added to the C1Ribbon control through the code using the **RibbonSeparator ('RibbonSeparator クラス' in the on-line documentation)** class. This is depicted in the code snippet below:

- **Visual Basic**

```
' リボン項目の区切り記号を追加します
Dim separatorItem As RibbonSeparator = New RibbonSeparator()
formatGroup.Items.Add(separatorItem)
```

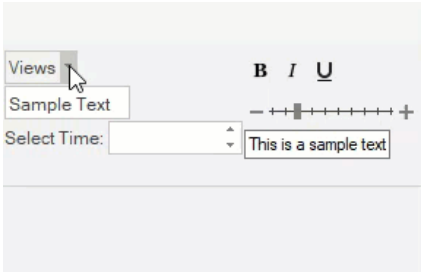
- **C#**

```
// リボン項目の区切り記号を追加します
RibbonSeparator separatorItem = new RibbonSeparator();
formatGroup.Items.Add(separatorItem);
```

スプリットボタン

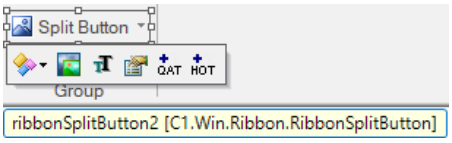
A split button is a combination of a regular button and a drop-down list. The split button is useful in scenarios where a user has to combine a set of variations of the same command.

The **Split Button** element is shown in the GIF below:



Adding SplitButton at Design-Time

The **Ribbon SplitButton** can be added at design-time using the **Ribbon Group Floating Toolbar** or **RibbonGroup Items Collection Editor**. Also, you can customize the look of the Ribbon SplitButton using the **Ribbon SplitButton Floating ToolBar** or by editing the properties in the **Properties** Window. For more information about floating toolbars, refer this topic. You can also add items to SplitButton using the **RibbonSplitButton Items Collection Editor**. Refer this topic for more Collection Editors.



Adding SplitButton through Code

A split button can also be added to the **C1Ribbon** control through the code using the **RibbonSplitButton ('RibbonSplitButton クラス' in the on-line documentation)** class. This is depicted in the code below:

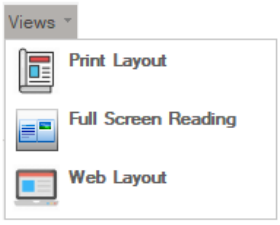
- **Visual Basic**

```
' Add RibbonSplitButton
Dim splitButton As RibbonSplitButton = New RibbonSplitButton("Views")
Dim printLayout As RibbonButton = New RibbonButton("Print Layout", Image.FromFile("images\printlayout.png"))
Dim fullScreenLayout As RibbonButton = New RibbonButton("Full Screen Reading", Image.FromFile("images\fullscreen.png"))
Dim webLayout As RibbonButton = New RibbonButton("Web Layout", Image.FromFile("images\weblayout.png"))
splitButton.Items.Add(printLayout)
splitButton.Items.Add(fullScreenLayout)
splitButton.Items.Add(webLayout)
formatGroup.Items.Add(splitButton)
'Specify the size of the items in dropdown menu
splitButton.PreferredItemSize = ItemSize.Auto
```

- **C#**

```
// Add Ribbon Split Button
RibbonSplitButton splitButton = new RibbonSplitButton("Views");
RibbonButton printLayout = new RibbonButton("Print Layout", Image.FromFile(@"images\printlayout.png"));
RibbonButton fullScreenLayout = new RibbonButton("Full Screen Reading", Image.FromFile(@"images\fullscreen.png"));
RibbonButton webLayout = new RibbonButton("Web Layout", Image.FromFile(@"images\weblayout.png"));
splitButton.Items.Add(printLayout);
splitButton.Items.Add(fullScreenLayout);
splitButton.Items.Add(webLayout);
formatGroup.Items.Add(splitButton);
// Specify the size of the items in dropdown menu
splitButton.PreferredItemSize = ItemSize.Auto;
```

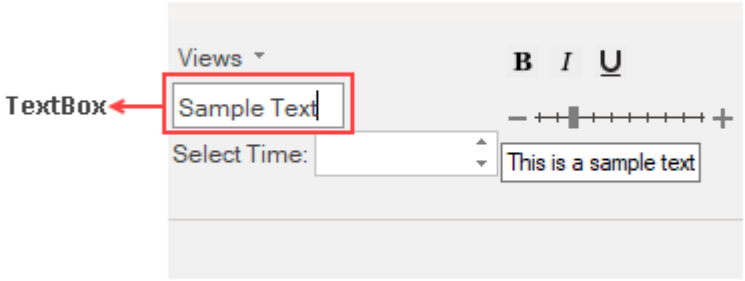
A user can specify the item size in the dropdown menu by specifying the **PreferredItemSize** property. The image below depicts the SplitButton item in the ribbon control when the **PreferredItemSize** property is set to Large.



テキストボックス

A TextBox enables the user to input text information in the application.

The **Ribbon TextBox** item is shown in the image below:



The TextBox can be added at design-time using the Floating Toolbar or Collection Editor. Refer this topic for detailed information. A TextBox can also be added to the C1Ribbon control through the code using the **RibbonTextBox ('RibbonTextBox クラス' in the on-line documentation)** class. This is depicted in the code below:

- **Visual Basic**

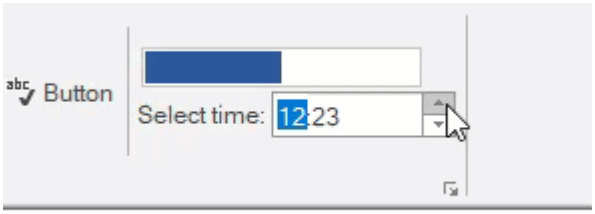
```
' TextBoxを追加します
Dim textBox As RibbonTextBox = New RibbonTextBox()
textBox.Text = "Sample Text"
formatGroup.Items.Add(textBox)
```

- **C#**

```
// TextBoxを追加します
RibbonTextBox textBox = new RibbonTextBox();
textBox.Text = "Sample Text";
formatGroup.Items.Add(textBox);
```

時刻ピッカー

The Time Picker works like a numeric box with increment and decrement buttons. It enables the user to choose a specific time or enter a specific time in the numeric box.



The Time Picker can be added through the designer using the Floating Toolbar or Collection Editor. Refer this topic for detailed information. A TimePicker can also be added to the C1Ribbon control through the code using the **RibbonTimePicker ('RibbonTimePicker クラス' in the on-line documentation)** class. This is depicted in the code below:

- **Visual Basic**

```
' TimePickerを追加します
Dim timePicker As RibbonTimePicker = New RibbonTimePicker()
timePicker.Label = "Select Time:"
formatGroup.Items.Add(timePicker)
```

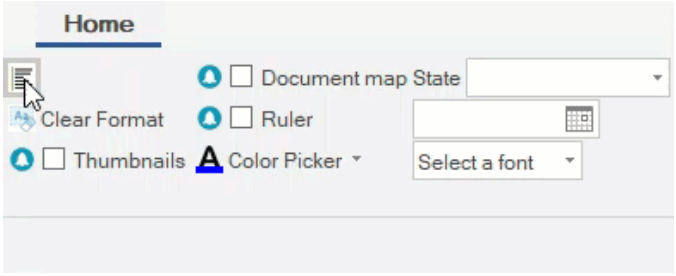
- **C#**

```
// TimePickerを追加します
RibbonTimePicker timePicker = new RibbonTimePicker();
timePicker.Label = "Select Time:";
formatGroup.Items.Add(timePicker);
```

トグルボタン

A toggle button is a command button that allows the user to switch between two states. When a user clicks on a toggle button, it switches to the pressed/active state and on consecutive clicking, it returns back to the unpressed/inactive state.

The GIF below shows a toggle button with tooltip:



The **Toggle Button** can be added at design-time using the Floating Toolbar or Collection Editor. Refer this topic for detailed information. A toggle button can also be added to the C1Ribbon control through the code using the **RibbonToggleButton ('RibbonToggleButton クラス' in the on-line documentation)** class. This is depicted in the code below.

- **Visual Basic**

```
' フォーマットグループにトグルボタンを追加します
Dim leftAlign As RibbonToggleButton = New RibbonToggleButton(Image.FromFile("images\align_left.png"))
leftAlign.ToolTip = "Align your content with left margin."
formatGroup.Items.Add(leftAlign)
```

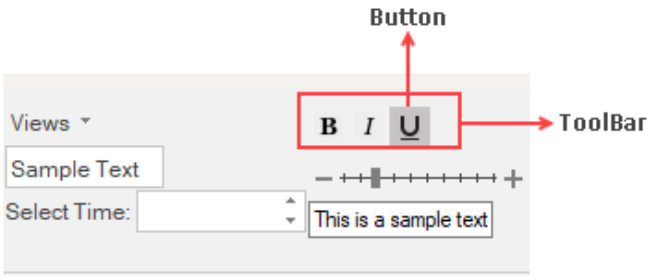
- **C#**

```
// フォーマットグループにトグルボタンを追加します
RibbonToggleButton leftAlign = new RibbonToggleButton(Image.FromFile(@"images\align_left.png"));
leftAlign.ToolTip = "Align your content with left margin.";
formatGroup.Items.Add(leftAlign);
```

ツールバー

A toolbar comprises of a row of clickable items that can perform different tasks. The **Ribbon ToolBar** provides easy access to the most frequently-used functions.

The image below shows a ToolBar with **Bold**, **Italics** and **Underline** buttons in the ribbon application.



The ToolBar can be added at design-time using the Floating Toolbar or Collection Editor. Refer this topic for detailed information. A toolbar can also be added programmatically to the Ribbon control using the **RibbonToolBar ('RibbonToolBar クラス' in the on-line documentation)** class. This is depicted in the code below:

- **Visual Basic**

```
' ツールバーを追加します
Dim ribbonToolBar As RibbonToolBar = New RibbonToolBar()
Dim boldButton As RibbonButton = New RibbonButton(Image.FromFile("images\bold.png"))
Dim italicButton As RibbonButton = New RibbonButton(Image.FromFile("images\italic.png"))
Dim underlineButton As RibbonButton = New RibbonButton(Image.FromFile("images\underline.png"))
ribbonToolBar.Items.Add(boldButton)
ribbonToolBar.Items.Add(italicButton)
ribbonToolBar.Items.Add(underlineButton)
formatGroup.Items.Add(ribbonToolBar)
```

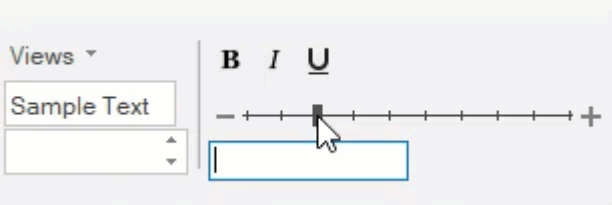
- **C#**

```
// ツールバーを追加します
RibbonToolBar ribbonToolBar = new RibbonToolBar();
RibbonButton boldButton = new RibbonButton(Image.FromFile(@"images\bold.png"));
RibbonButton italicButton = new RibbonButton(Image.FromFile(@"images\italic.png"));
RibbonButton underlineButton = new RibbonButton(Image.FromFile(@"images\underline.png"));
ribbonToolBar.Items.Add(boldButton);
ribbonToolBar.Items.Add(italicButton);
ribbonToolBar.Items.Add(underlineButton);
formatGroup.Items.Add(ribbonToolBar);
```

トラックバー

The Track Bar is a horizontal slider that enables the user to select values on a bar by moving the slider. The tick marks are spaced at regular intervals called the tick frequency. The user can set the Maximum and Minimum values on the TrackBar.

The GIF below shows the **Ribbon TrackBar** in a ribbon application.



The Track Bar can be added via the designer using Floating Toolbar or Collection Editor. Refer this topic for detailed information. The TrackBar can also be added programmatically to the Ribbon control using the **RibbonTrackBar ('RibbonTrackBar クラス' in the on-line documentation)** class. This is depicted in the code below:

- **Visual Basic**

```
' TrackBarを追加します
Dim trackBar As RibbonTrackBar = New RibbonTrackBar()
trackBar.Minimum = 10
trackBar.Maximum = 100
trackBar.Value = 30
trackBar.TickFrequency = 10
formatGroup.Items.Add(trackBar)
```

- **C#**

```
// TrackBarを追加します
RibbonTrackBar trackBar = new RibbonTrackBar();
trackBar.Minimum = 10;
trackBar.Maximum = 100;
trackBar.Value = 30;
trackBar.TickFrequency = 10;
trackBar.Width = 200;
formatGroup.Items.Add(trackBar);
```

コントロールホスト

The **Control Host** item enables the user to add a hosted control in the RibbonGroup of the **C1Ribbon** control. A ControlHost item can be added both through the designer as well as code. In both the cases, the user has to programmatically configure the hosted control in the MainForm. Let's say a user wants to add a TextBox control. For this, the user needs to create a new **TextBoxHost** class that inherits the **RibbonControlHost** element. Likewise, in order to add a ComboBox control, the user can create a new **ComboBoxHost** class that inherits the **RibbonControlHost** element.

The following sections cover in detail about the configuration of a **ControlHost** in the **C1Ribbon** control via both the designer and code.

By Designer

Complete the following steps to add a standard **TextBox** control to a Ribbon group via the designer:

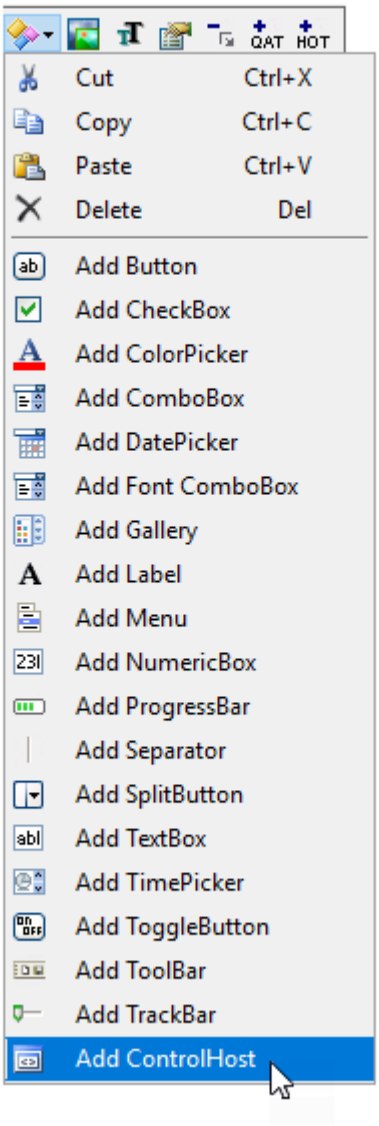
1. Open the **MainRibbonForm** to view the Ribbon Form, and select **View Code** to open code view.
2. Add the following code to your project to create a new **TextBoxHost** class that inherits the **RibbonControlHost** element:
    - **Visual Basic**

```
    Public Sub New()
    MyBase.New(New System.Windows.Forms.TextBox())
    MyBase.Text = "This is a sample text."
End Sub
```

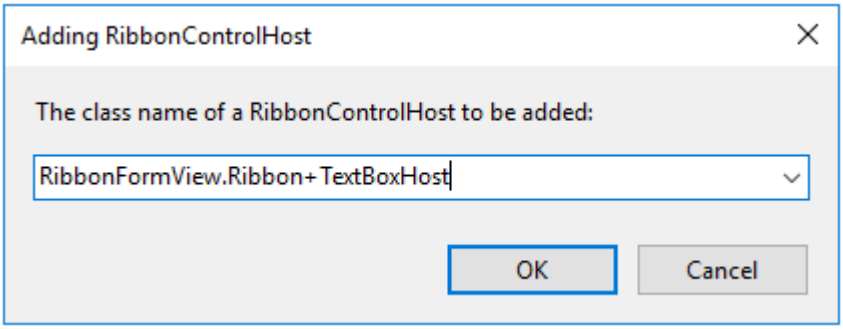
    - **C#**

```
public class TextBoxHost : C1.Win.Ribbon.RibbonControlHost
{
    public TextBoxHost() : base(new System.Windows.Forms.TextBox())
    {
        base.Text = "This is a sample text.";
    }
}
```

3. Build and close your project, and then return to Design view.
4. Open the floating toolbar of the Group to which you want to add the ControlHost item.
5. Click the **Actions** button, and select **Add ControlHost**.



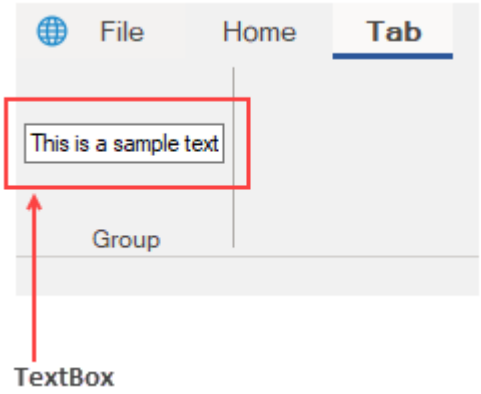
6. The **Adding RibbonControlHost** dialog box will launch. The dialog box will request the class name of the RibbonControlHost. Enter the name of the control host in the **Adding RibbonControlHost** dialog box in this format "ProjectName.FormName+TextBoxHost" replacing ProjectName and FormName with the names of your project and form.



> Note: When you again add a TextBoxHost element, you won't have to type its name. The name of this class will be available in the drop-down list in the Adding RibbonControlHost dialog box.

7. Click **OK** to close the **Adding RibbonControlHost** dialog box.

Notice that the **TextBox** control now appears in the Ribbon group.



By Code

The **RibbonControlHost** item can be added programmatically with the **RibbonControlHost ('RibbonControlHost クラス' in the on-line documentation)** class as shown in the code below:

- **Visual Basic**

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    Dim textboxHost As RibbonControlHost = New RibbonControlHost()
    textboxHost = New AddRibbonItems.TextBoxHost()
    formatGroup.Items.Add(textboxHost)
End Sub

Public Class TextBoxHost
    Inherits C1.Win.Ribbon.RibbonControlHost

    Public Sub New()
        MyBase.New(New System.Windows.Forms.TextBox())
        MyBase.Text = "This is a sample text."
    End Sub
```

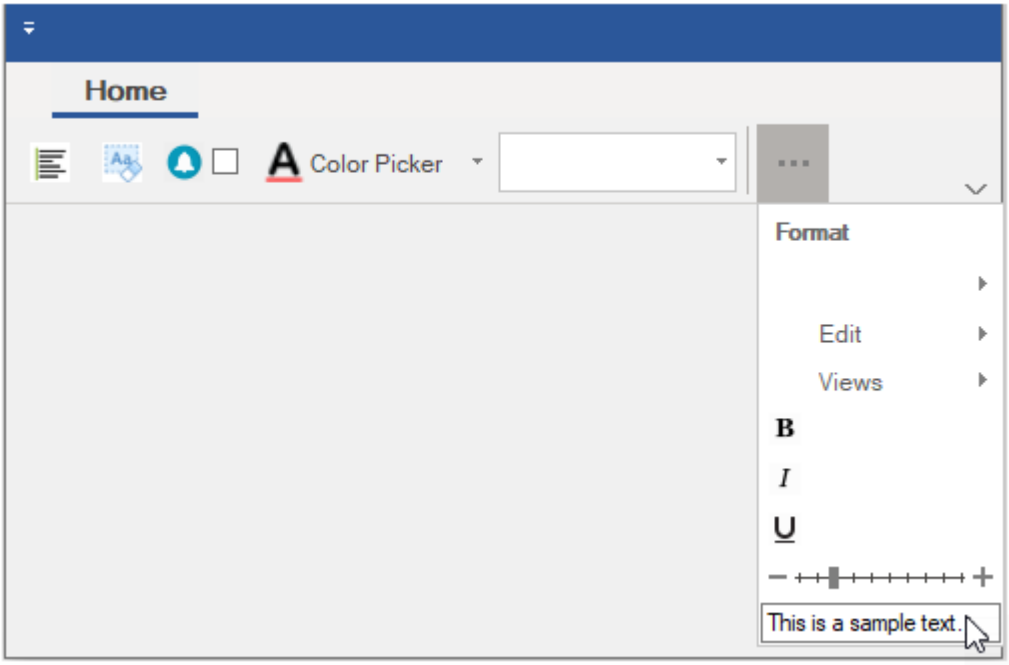
- **C#**

```
private void Form1_Load(object sender, EventArgs e)
        {

            //ControlHost項目を追加します
            RibbonControlHost textboxHost = new RibbonControlHost();
        textboxHost = new AddRibbonItems.TextBoxHost();
        formatGroup.Items.Add(textboxHost);

        }
    }

public class TextBoxHost : C1.Win.Ribbon.RibbonControlHost
{
    public TextBoxHost(): base(new System.Windows.Forms.TextBox())
    {
        base.Text = "This is a sample text.";
    }
}
```

Availability in the Dropdown

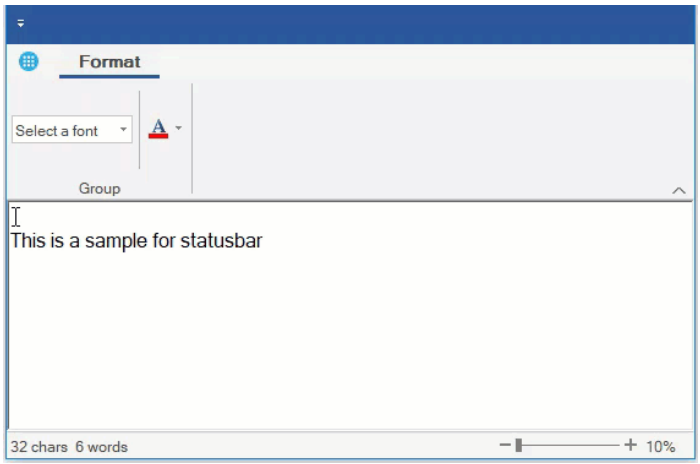
When the Ribbon is switched to its Simplified state, you can easily view the RibbonControlHost using the dropdown

button as shown in the snapshot below.



ステータスバー

The **C1StatusBar** control appears like a horizontal bar at the bottom of the Forms window. The StatusBar can display various kinds of status information in an application window.



The **C1StatusBar** control is a separate control and can be added at design-time from the Toolbox. When dropped on the Forms, **C1StatusBar** gets docked at the bottom. Items can be added to the left and right pane of the StatusBar by using the floating toolbar in context menu or the collection editors in the **Properties** window. For more information about adding items through the designer, refer this topic.

The StatusBar control can also be added to the form programmatically. The items can be added to the left pane and right pane of the StatusBar using the **LeftPaneItems ('LeftPaneItems プロパティ' in the on-line documentation)** and **RightPaneItems ('RightPaneItems プロパティ' in the on-line documentation)** properties of the **C1StatusBar ('C1StatusBar クラス' in the on-line documentation)** class.

- **Visual Basic**

```
Public Class Form1

    Private statusBar As C1StatusBar
    Private _percents As Integer() = New Integer() {10, 20, 30, 40, 50, 60,
        70, 80, 90, 100, 120, 150,
        200, 250, 300, 400, 700, 1000}

    Public Sub New()
        InitializeComponent()
        'ステータスバーをリボンコントロールに追加します
        AddStatusBar(Me)
    End Sub

    Public Sub AddStatusBar(form1 As Form1)
        'ステータスバーを作成して追加します
        statusBar = New C1StatusBar()
        statusBar.Width = 150

        'パーセンテージ値を表示するボタンを作成します
        Dim zoomButton As New RibbonButton(Nothing, Image.FromFile("images\zoomButton.png"))
        Dim wordcountLabel As New RibbonLabel()
        wordcountLabel.Text = "0 words"
```

```
        Dim charcountLabel As New RibbonLabel()
        charcountLabel.Text = "0 chars"

        'TextChangedイベントハンドラーを追加します
        AddHandler RichTextBox1.TextChanged, AddressOf RichTextBox1_TextChanged

        'リボントラックバーを作成します
        Dim zoomTrackBar As New RibbonTrackBar()
        zoomTrackBar.Minimum = 0
        zoomTrackBar.Maximum = 17
        zoomTrackBar.StepFrequency = 1

        'イベントハンドラーを追加します
        AddHandler zoomTrackBar.ValueChanged, AddressOf ZoomTrackBar1_ValueChanged

        'アイテムを左右のパネルに追加します
        statusBar.RightPaneItems.Add(zoomTrackBar)
        statusBar.RightPaneItems.Add(New RibbonLabel("10%"))
        statusBar.LeftPaneItems.Add(wordcountLabel)
        statusBar.LeftPaneItems.Add(charcountLabel)

        'ステータスバーをメインフォームに追加します
        Me.Controls.Add(statusBar)
    End Sub

    Private Sub ZoomTrackBar1_ValueChanged(sender As Object, e As EventArgs)
        Dim zoomBar As RibbonTrackBar = DirectCast(sender, RibbonTrackBar)
        Dim index = zoomBar.Value
        Dim zf As Integer = _percents(index)
        Dim zoomLabel = DirectCast(statusBar.RightPaneItems(1), RibbonLabel)
        zoomLabel.Text = zf.ToString() & "%"
        Me.RichTextBox1.ZoomFactor = zf / 100.0F
    End Sub

    Private Sub RichTextBox1_TextChanged(sender As Object, e As EventArgs)
        Dim charcountLabel As RibbonLabel = DirectCast(statusBar.LeftPaneItems(0), RibbonLabel)
        Dim wordcountLabel As RibbonLabel = DirectCast(statusBar.LeftPaneItems(1), RibbonLabel)
        charcountLabel.Text = RichTextBox1.TextLength.ToString() & " chars"
        wordcountLabel.Text = RichTextBox1.Text.Split(New Char() {" "c}, StringSplitOptions.RemoveEmptyEntries).Length.ToString() & " words"
    End Sub

End Class
```

- **C#**

```
public partial class Form1 : Form
{
    C1StatusBar statusBar;
    int[] _percents = new int[] { 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 120, 150, 200, 250, 300, 400, 700, 1000 };
    public Form1()
    {
        InitializeComponent();
        //ステータスバーをリボンコントロールに追加します
        AddStatusBar(this);
    }

    public void AddStatusBar(Form1 form1)
    {

        //ステータスバーを作成して追加します
        statusBar = new C1StatusBar();
        statusBar.Width = 150;

        //パーセンテージ値を表示するボタンを作成します
        RibbonButton zoomButton = new RibbonButton(null, Image.FromFile("images\\zoomButton.png"));
        RibbonLabel wordcountLabel = new RibbonLabel();
        wordcountLabel.Text = "0 words";
        RibbonLabel charcountLabel = new RibbonLabel();
        charcountLabel.Text = "0 chars";

        //TextChangedイベントハンドラーを追加します
        richTextBox1.TextChanged += RichTextBox1_TextChanged;

        //リボントラックバーを作成します
        RibbonTrackBar zoomTrackBar = new RibbonTrackBar();
        zoomTrackBar.Minimum = 0;
        zoomTrackBar.Maximum = 17;
        zoomTrackBar.StepFrequency = 1;

        //イベントハンドラーを追加します
        zoomTrackBar.ValueChanged += ZoomTrackBar1_ValueChanged;

        //アイテムを左右のパネルに追加します
        statusBar.RightPaneItems.Add(zoomTrackBar);
        statusBar.RightPaneItems.Add(new RibbonLabel("10%"));
        statusBar.LeftPaneItems.Add(wordcountLabel);
        statusBar.LeftPaneItems.Add(charcountLabel);

        //ステータスバーをメインフォームに追加します
        this.Controls.Add(statusBar);

    }

    private void ZoomTrackBar1_ValueChanged(object sender, EventArgs e)
    {
        RibbonTrackBar zoomBar = (RibbonTrackBar)sender;
        var index = zoomBar.Value;
        int zf = _percents[index];
        var zoomLabel = (RibbonLabel)statusBar.RightPaneItems[1];
        zoomLabel.Text = zf.ToString() + "%";
        this.richTextBox1.ZoomFactor = zf / 100f;
    }
    private void RichTextBox1_TextChanged(object sender, EventArgs e)
    {
```

```
        RibbonLabel charcountLabel = (RibbonLabel)statusBar.LeftPaneItems[0];
        RibbonLabel wordcountLabel = (RibbonLabel)statusBar.LeftPaneItems[1];
        charcountLabel.Text = richTextBox1.TextLength.ToString() + " chars";
        wordcountLabel.Text = richTextBox1.Text.Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries).Length.ToString() + " words";
    }

}
```

設計時のサポート

The Ribbon for WinForms provides visual editing to make it easier for a user to create Ribbon in their applications. You can make changes to Ribbon by using **Smart Tags**, **Floating Toolbars** or **Collection Editors**. The following topics will walk you through **C1Ribbon** Design-Time support.

Smart Tags

Application Menu

Backstage View

Quick Access Toolbar

Configuration Toolbar

Contextual Tab Group

Ribbon Tab

Ribbon Group

RibbonGroup Items

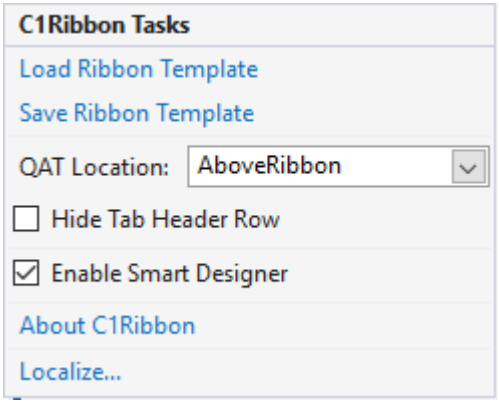
StatusBar

スマートタグ

The **C1Ribbon** and **C1StatusBar** controls provide quick and easy access to their common properties through their smart tags.

Ribbon Smart Tag

To access the Ribbon Tasks menu, click on the smart tag (▶) in the upper-right corner of the Ribbon control. This will open the **C1Ribbon Tasks** menu.



The C1Ribbon Tasks menu can perform the following operations:

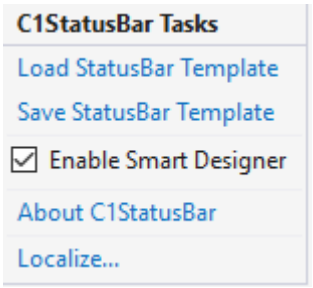
- **Load Ribbon Template**
  It opens the Load Ribbon Template dialog box where you can import an XML file that contains the pre-formatted Ribbon.
- **Save Ribbon Template**
  It opens Save Ribbon Template dialog box where you can save the Ribbon layout as an XML file.
- **QAT Location**
  It allows you to select the location of the Quick Access Toolbar in relation to the Ribbon.
- **Enable Smart Designer**
  Selecting or deselecting the Enable Smart Designer check box turns on or off the smart designer functionality.
- **About C1Ribbon**

It displays a dialog box, which is helpful in finding the version number of C1Ribbon and online resources.

- **Localize**
  Opens the Localize dialog box, from where you can add user-defined localization for run-time string resources.

StatusBar Smart Tag

To access the StatusBar Tasks menu, click on the smart tag (▶) in the upper-right corner of the StatusBar control. This will open the C1StatusBar Tasks menu.



The C1StatusBar Tasks menu can perform the following operations:

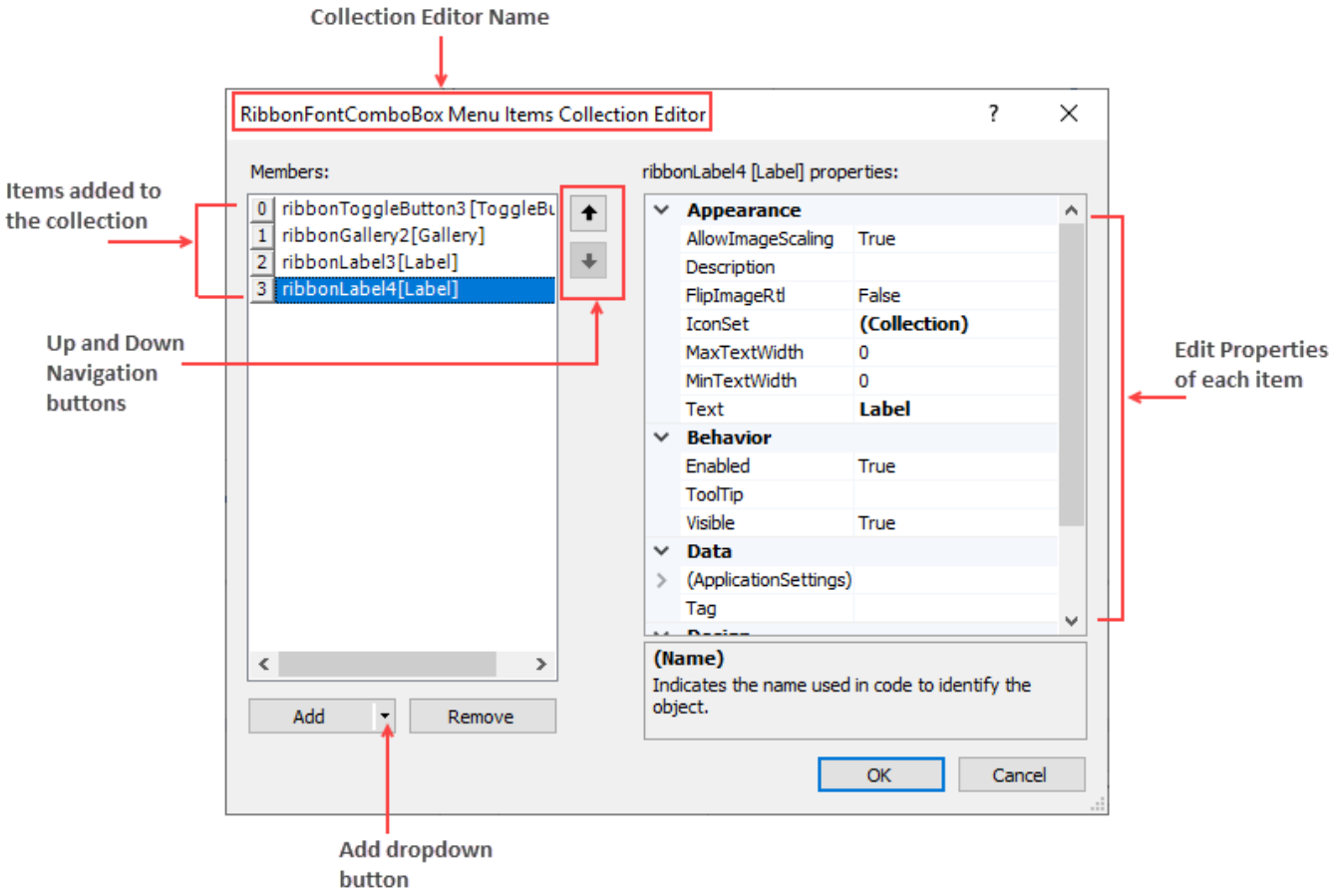
- **Load StatusBar Template**
  It opens the Load StatusBar Template dialog box where you can import an XML file that contains the pre-formatted status bar.
- **Save StatusBar Template**
  It opens Save StatusBar Template dialog box where you can save the status bar layout as an XML file.
- **Enable Smart Designer**
  Selecting or deselecting the Enable Smart Designer check box turns on or off the smart designer functionality respectively.
- **About C1StatusBar**
  It displays a dialog box, which is helpful in finding the version number of C1Ribbon and online resources.
- **Localize**
  Opens the Localize dialog box, from where you can add user-defined localization for run-time string resources.

コレクション エディター

Collection Editors are used to add, remove or edit items in a collection at design-time. A Collection Editor can be launched by selecting the Ribbon element on the Form and then clicking the ellipsis next to **Items**, **MenuItems**, **LeftPaneItems**, **RightPaneItems**, **BottomPaneItems** or **LeftBottomPaneItems** properties in their respective **Properties** window.

The text label at the top of the Collection Editor Dialog Box displays the name of Collection Editor. The Collection Editor dialog box usually comprises two sections: left and right sections. The Left part of the Collection Editor contains the **Add** and **Remove** button that can be used to add and remove items. The added items gets displayed in the **Members** section of the Collection Editor. You can click on each item added in the Members sections to view the respective item properties on the right pane. The Up and Down arrow buttons are used to navigate up and down the Members.

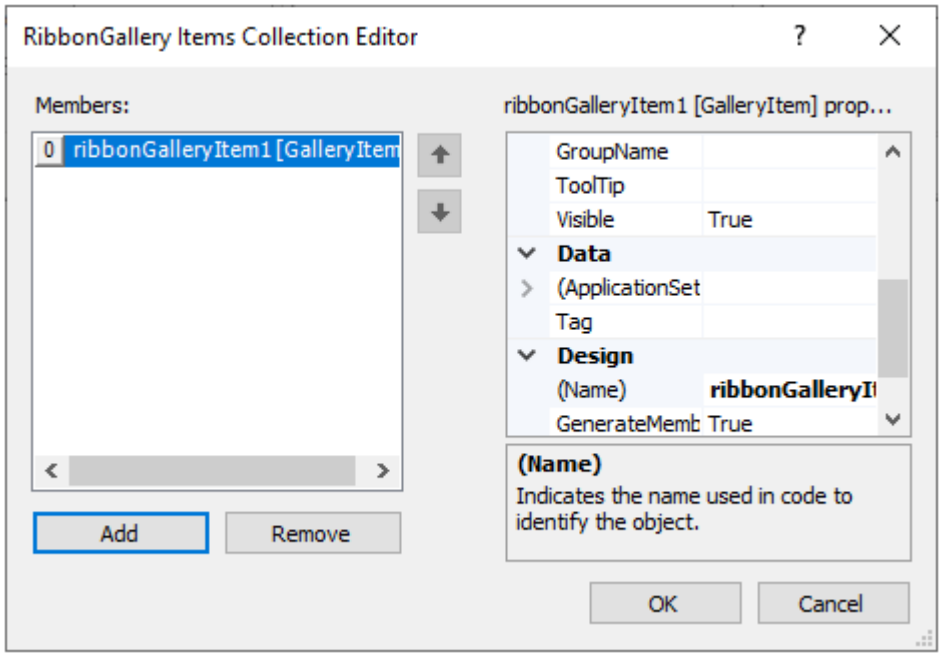
The image below depicts the **RibbonFontComboBox Menu Items Collection Editor**.



Types of Collection Editors

Collection editors are grouped into three major types: Items Collection Editors, Menu Items Collection Editors and PaneItems Collection Editors.

- **Items Collection Editors**: It is used to add items to a particular Ribbon element. The image below depicts an example of the Items Collection Editor.

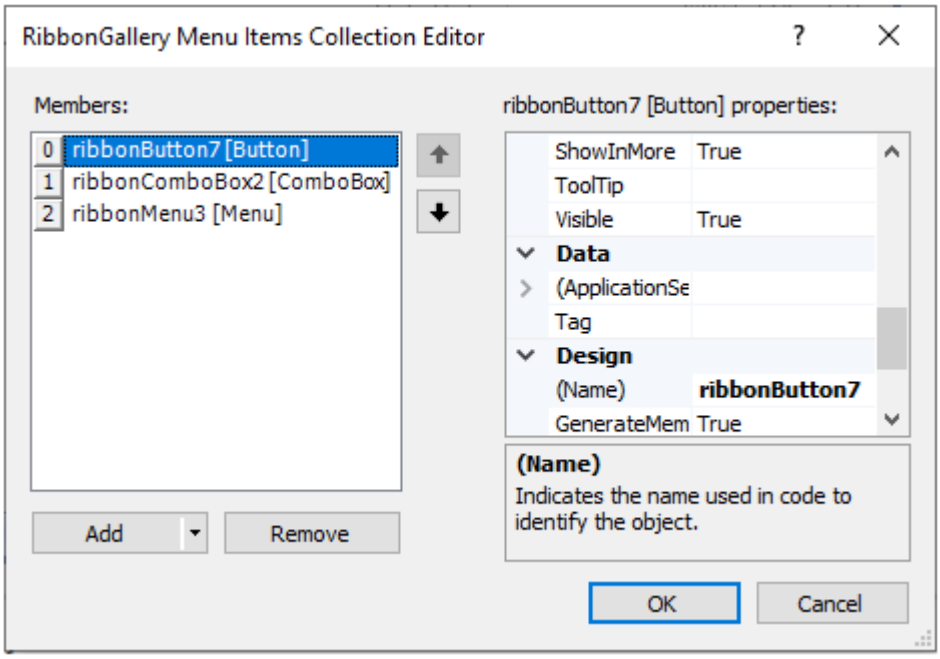


Let's say a user wants to add and customize item(s) in the Gallery using **RibbonGallery Items Collection**

Editor. Follow these steps to add items to the Items collection editor:

1. Click the RibbonGallery element on the Form at design time to activate it.
2. In the Properties window, click on the ellipsis next to the **Items** property. The **RibbonGallery Items Collection Editor** appears.
3. Click the **Add** button to add gallery item(s) to the Gallery. Click the **OK** button.

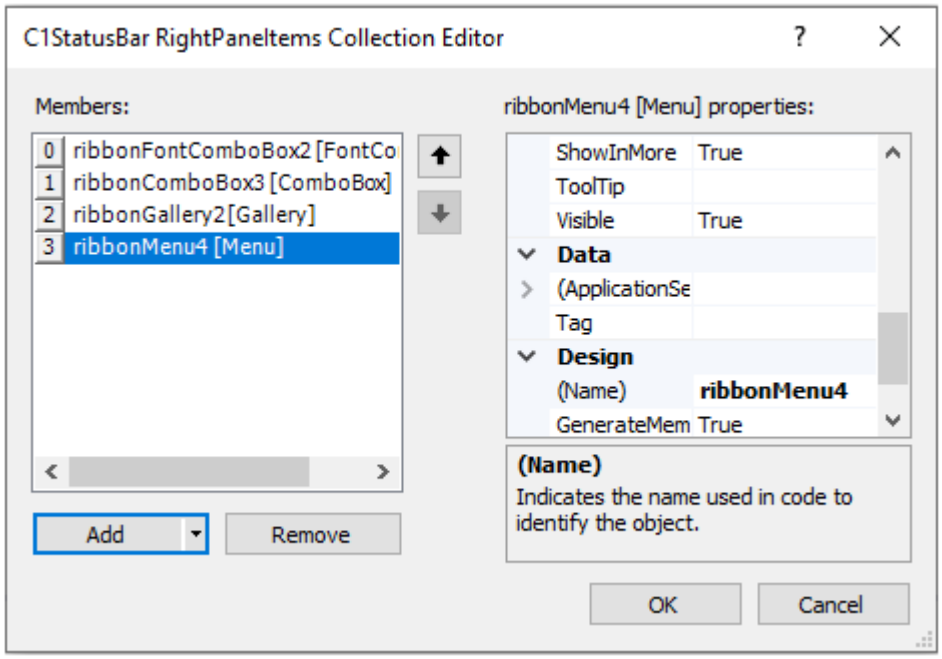
- **Menu Items Collection Editors:** It is used to add menu items to the dropdown portion of a Ribbon Element. For example, menu items can be added to Gallery using the **RibbonGallery MenuItems Collection Editor**. The image below depicts an example for Menu Items Collection Editor.



Let's say a user wants to add and customize menu item(s) in the Gallery using **RibbonGallery MenuItems Collection Editor.** Follow the steps below to add items to the collection editor:

1. Click the RibbonGallery element on the Form at design time to activate it.
2. In the Properties window, click on the ellipsis next to the **MenuItems** property. The **RibbonGallery MenuItems Collection Editor** appears.
3. Click the Add drop-down button to add menu items to the Gallery.

- **PaneItems Collection Editors:** Certain Ribbon Elements contain left, right, bottom or left bottom panes. Few examples are Application Menu, Backstage View and StatusBar. Items can also be added to panes at design time using **PaneItems Collection Editors**.
  The Application Menu contains Left Pane, Right Pane and Bottom Pane, while the Backstage View tab contains Left Bottom Pane and Left Pane. Likewise, the C1StatusBar contains Left Pane and Right Pane. The image below depicts an example for PaneItems Collection Editor.



Let's say a user wants to add and customize item(s) in the right pane of the StatusBar using **C1StatusBar RightPaneItems Collection Editor.** Follow these steps to add items to the collection editor:

1. Select the **C1StatusBar** on the Form at design time.
2. In the **Properties** window, click on the ellipsis next to the RightPaneItems. The RibbonGallery MenuItems Collection Editor appears.
3. Click the Add drop-down button to add items to the Right Pane of the C1StatusBar.

Refer the table below to get an in-depth idea about the Collection Editors associated with different elements in the C1Ribbon control.

Elements	Collection Editors	Collection Editor Types
C1Ribbon	✓	C1Ribbon has 13 types of Collection Editors: • RibbonApplicationMenu LeftPaneItems Collection Editor • RibbonApplicationMenu RightPaneItems Collection Editor • RibbonApplicationMenu BottomPaneItems Collection Editor • RibbonBottomToolBar Items Collection Editor • RibbonConfigToolBar Items Collection Editor • RibbonMenu Items Collection Editor • RibbonContextualTabGroup Collection Editor • QAT Items Collection Editor • QAT MenuItems Collection Editor • RibbonTopToolBar Items Collection Editor • Backstage View LeftBottomPaneItems Collection Editor • Backstage View LeftPaneItems Collection Editor

		• RibbonTab Collection Editor
Application Menu	✓	Application Menu has 3 types of Collection Editors: • RibbonApplicationMenu LeftPaneItems Collection Editor • RibbonApplicationMenu RightPaneItems Collection Editor • RibbonApplicationMenu BottomPaneItems Collection Editor
Backstage View		BackStage View has 2 types of Collection Editors: • Backstage View LeftBottomPaneItems Collection Editor • Backstage View LeftPaneItems Collection Editor
Configuration Toolbar	✓	RibbonConfigToolBar Items Collection Editor
Contextual Tab Group	✓	RibbonTab Collection Editor
Quick Access Toolbar	✓	The QAT has 2 types of Collection Editors: • QAT Items Collection Editor • QAT MenuItems Collection Editor
Ribbon Group	✓	RibbonGroup Items Collection Editor
Ribbon Tab	✓	RibbonGroup Collection Editor
StatusBar	✓	StatusBar has 2 types of Collection Editors: • C1StatusBar LeftPaneItems Collection Editor • C1StatusBar RightPaneItems Collection Editor
Button	✗	
CheckBox	✗	
ColorPicker	✗	
ComboBox	✓	ComboBox has 2 types of Collection Editors: • RibbonComboBox Items Collection Editor • RibbonComboBox MenuItems Collection Editor
ControlHost	✗	
Date Picker	✗	

Font ComboBox	✓	RibbonFontComboBox Menu Items Collection Editor
Gallery	✓	Gallery has 2 types of Collection Editors: • RibbonGallery Items Collection Editor • RibbonGallery MenuItems Collection Editor
Label	✗	
Menu	✓	RibbonMenu Items Collection Editor
NumericBox	✗	
ProgressBar	✗	
Separator	✗	
SplitButton	✓	RibbonSplitButton Items Collection Editor
TextBox	✗	
Time Picker	✗	
Toggle Button	✗	
ToolBar	✓	RibbonToolBar Items Collection Editor
TrackBar	✗	

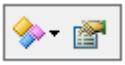
フリー ツール バー

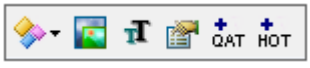
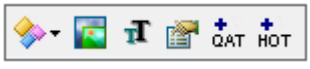
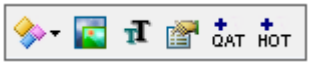
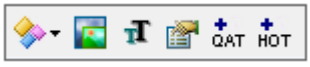
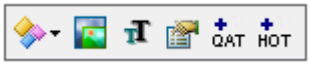
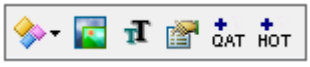
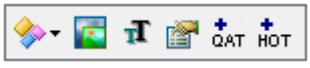
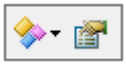
The **Floating Toolbar** is a part of the Ribbon **Smart Designer**, which allows the user to customize different types of Ribbon Elements at design time. **C1Ribbon** provides visual editing to make it easier to create a Ribbon at design time. The user can set properties directly on the form using the Smart Designer Floating Toolbar.

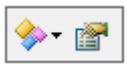
Hovering the mouse over an item in the C1Ribbon smart designer in the form displays the ribbon element name. This is depicted in the GIF below:

Floating Toolbar Types

Each element in the Ribbon control has a floating toolbar. This is elaborated in the table below:

Elements	Floating Toolbar
C1Ribbon	
Application Menu	
BackstageView	
Quick Access Toolbar	
Configuration Toolbar	

Contextual Tab Group	
Ribbon Tab	
Ribbon Group	
StatusBar	
Button	
CheckBox	
ColorPicker	
ComboBox	
ControlHost	
Date Picker	
Font ComboBox	
Gallery	
Label	
Menu	
Numeric Box	
Progress Bar	
Separator	
SplitButton	
TextBox	
Time Picker	
Toggle Button	
Tool Bar	
Track Bar	

Top Toolbar	
Bottom Toolbar	

From the table, it is evident that many ribbon elements have the same type of floating toolbar. For example, ribbon items like QAT, Configuration Toolbar, Progress Bar, Tool Bar, Track Bar, Top Toolbar and Bottom Toolbar have the same type of floating toolbar.

Floating Toolbar Options

The different types of options available in the floating toolbars are elucidated below.

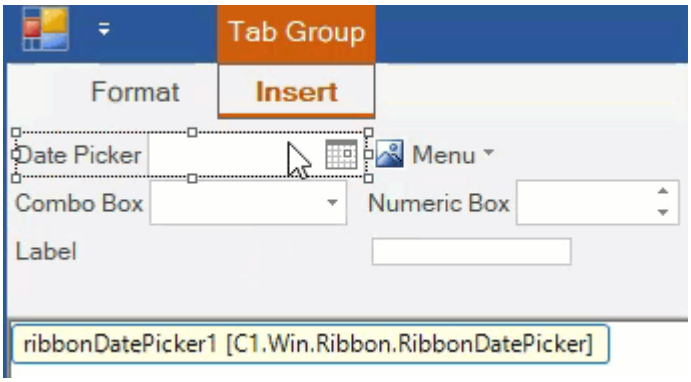
Floating Toolbar Options	Description
	**Actions:** It opens a dropdown menu that usually contains the **Cut**,**Copy**, **Paste** and **Delete**options. This lets the user cut, copy, paste or delete that item. Depending on the type of Ribbon Item, more options can appear in the **Actions** dropdown menu. For example, the Tab Floating Toolbar has an extra option **Add Group**, while the Group Floating Toolbar has options like **AddButton**, **Add CheckBoxes**, **Add TrackBar**, **Add ControlHost** etc. The Contextual Tab Group has an **Add Tab** option to add tabs. The Actions menu in the C1Ribbon floating toolbar differs from the rest items and has unique options like **Load Ribbon Template**, **Save Ribbon Template**, **Add Tab** and **Add Contextual Tab** Group.
	**Change Image**: Use the **Change Image** option to customize the appearance/icon of the item. The Change Image button now opens a window with four options specifying image sizes in advance. The user can conveniently choose from 16 x 16 px, 20 x 20 px, 32 x 32 px images or select a custom image. Clicking any one of the four options opens the **Select Image** editor. Refer the Icons topic to know more about the Select Image editor.
	**Text settings**: When the user clicks this option, the Text Setting dialog box appears. The user can add the **Text**, **ToolTip**, **Cue**, **Label**, **Description** etc. to the Ribbon element.
	**Miscellaneous settings:** The Miscellaneous Settings option allows the user to configure certain Properties associated with an element. Note that it can vary with Ribbon elements. For example, in case of Ribbon ComboBox, TextBox and FontComboBox, the Miscellaneous Settings dialog box allows the user to edit Properties like **Text Area Width**, **Max Text Length** and **Horizontal Alignment**. Similarly, the TrackBar and ProgressBar has a Miscellaneous Settings dialog Box with properties like **Maximum** and **Minimum**. Likewise, the Toggle Button Miscellaneous Settings contains the **ToggleGroupName** property.The Miscellaneous Settings option for other Ribbon elements contains check boxes to enable or disable the respective ribbon element.
	**Add Launcher Button / Remove Launcher Button:** This option allows to add or remove a dialog box launcher button to the Ribbon Group.
	**Add to Quick Access Toolbar:** This option allows the user to add the Ribbon element to QAT (Quick Access Toolbar).
	**Add to Hot Item List:** This option allows the user to add the Ribbon element to the Hot Item List.
	**Contextual Color Scheme:** It lets you change the color scheme of the Contextual Tab Group.

	**Localize:** The Localize option opens the Localize dialog box.
	**Hide/show Ribbon items**: It allows you to hide or show Ribbon items in the ribbon UI. This is available in the floating toolbar of C1Ribbon. Note that while adding more tabs or groups, the smart designer may hide some tabs or groups. But the Hide/Show Ribbon Items option lets the user get back to the first tab or group.

In-Place Caption Editing

An interesting feature of the smart designer is that it allows in-place text editing of the Ribbon element on the Form. Clicking on a Ribbon element launches the floating toolbar and a tooltip '**Click to edit item text/tabcaption**'. As directed in the tooltip, the user can edit the text by simply clicking on the text label with the mouse pointer.

This is illustrated in the GIF below:



リボンの使用

The **C1Ribbon** control is a collection of Tabs, Groups and Group Items. **Ribbon for WinForms** helps you to configure a feature-rich Ribbon UI. This section will walk you through the critical aspects of working with C1Ribbon:

Design-Time Support

Simplified Ribbon

TooltTip

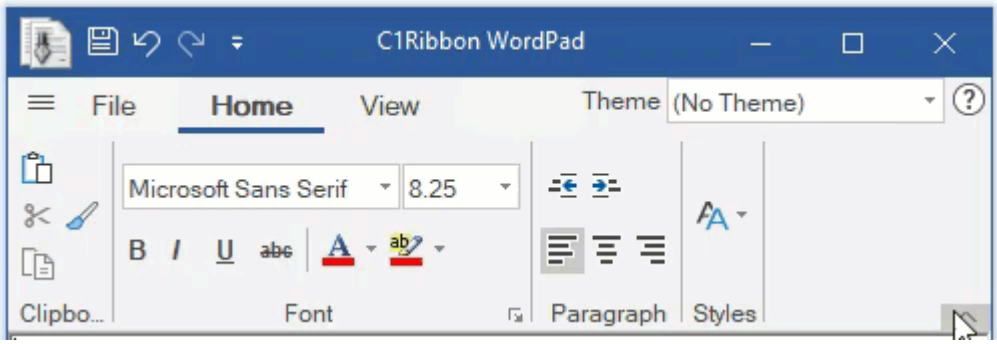
Themes

Icon

Keyboard Support

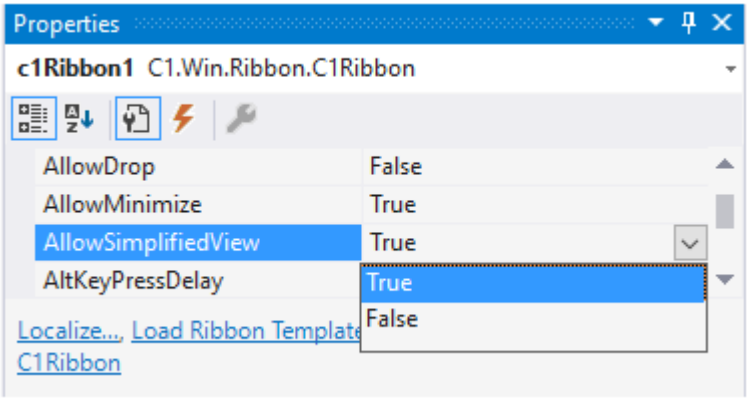
簡略化リボン

The **C1Ribbon** control allows a user to switch between simplified and full view of the Ribbon. The simplified view makes the ribbon more streamlined and occupy less space on the screen.



At runtime, this can be done using a chevron button located at the bottom right corner of the ribbon. By default, the simplified view feature is present in C1Ribbon, but a user can disable it using the **AllowSimplifiedView ('AllowSimplifiedView プロパティ' in the on-line documentation)** property of **C1Ribbon ('C1Ribbon クラス' in the on-line documentation)** class.

At design time, the user can disable the simplified view feature using the **AllowSimplifiedView** property of **C1Ribbon** in the **Properties** window.



This can be done programmatically as shown in the code snippet below:

- **Visual Basic**

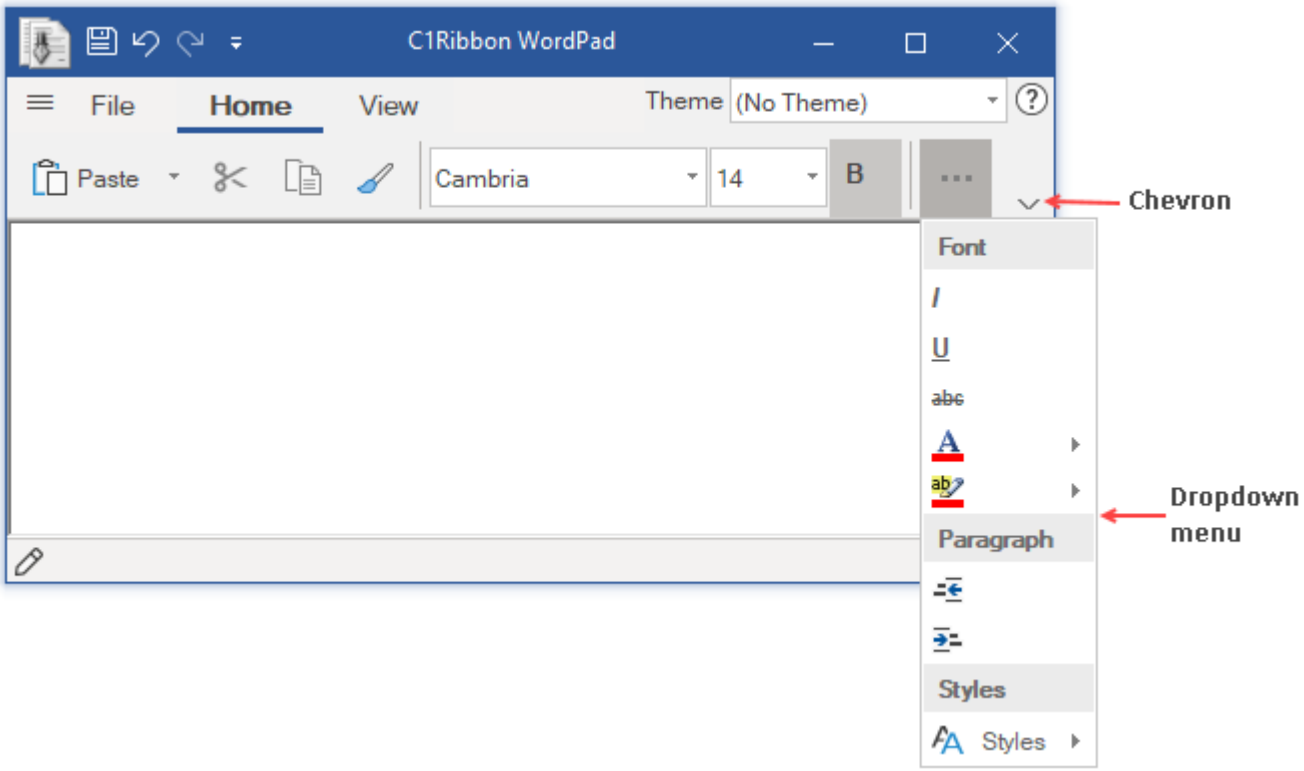
```
'簡略化ビューを無効にします
```

```
customRibbon.AllowSimplifiedView = False
```

- **C#**

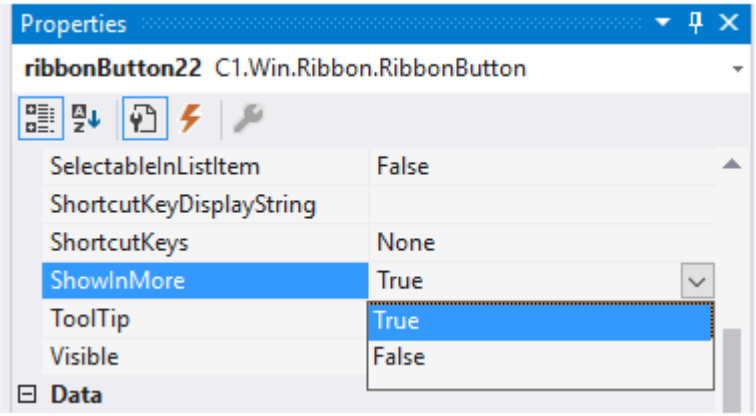
```
//簡略化ビューを無効にします
customRibbon.AllowSimplifiedView = false;
```

In the simplified view, some elements are shown in a single row, while some other are shown using the dropdown menu. This is illustrated in the image below.



Note that all the elements of the ribbon can not be shown in the drop-down part of the simplified view. For example, the user can specify not to show a button element in the dropdown using the **ShowInMore ('ShowInMore プロパティ' in the on-line documentation)** property of **RibbonButton ('RibbonButton クラス' in the on-line documentation)** class.

At design time, this can be done using the **ShowInMore** property of a ribbon item in the **Properties** window.



You can hide an element from the dropdown menu via code as shown below:

- **Visual Basic**

```
'簡略化リボンのドロップダウンメニューから要素を非表示にします
```

```
clearButton.ShowInMore = False
```

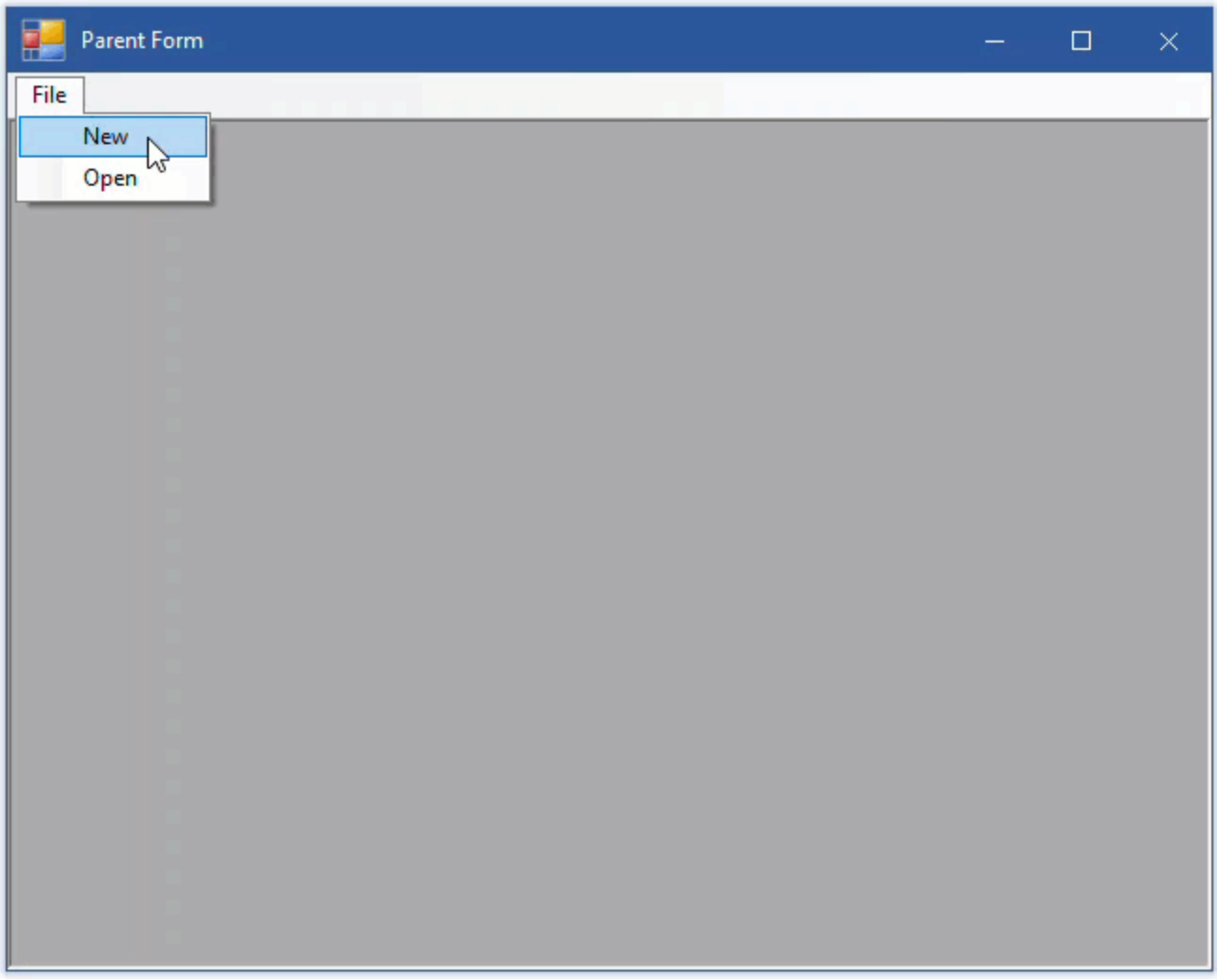
- **C#**

```
//簡略化リボンのドロップダウンメニューから要素を非表示にします
clearButton.ShowInMore = false;
```

MDI 子 RibbonForm

The **Ribbon** control supports **MDI (Multiple document Interface) Child Ribbon Forms**, which is quite useful in user interaction. This section illustrates how to create MDI Child forms using the Ribbon control.

The RibbonForm can be used in MDI scenarios. The MDI feature lets you share a single menu bar between all child windows, increasing the efficiency of screen space usage.



Creating MDI form

In order to create an MDI parent RibbonForm and child form, follow the steps below:

1. Create a RibbonForm.

C#

```
partial class Form1 : C1RibbonForm
{
    //...
```

```
}
```

2. Set up the MDI parent form by setting **Form.IsMdiContainer** property to true, and make the window maximized.

C# copyCode

```
public ParentForm()
{
    InitializeComponent();
    this.Text = "Parent Form";
    this.IsMdiContainer = true;
    this.WindowState = FormWindowState.Maximized;
}
```

3. Add tool strip menu items, 'New' and 'Open' buttons to the menu strip in the RibbonForm, such that on clicking each button a child form opens, NewForm and OpenForm (which, we will create in the next step).

4. Add the following code to create and display the MDI child form 'NewForm' and assign border size to the MDI Child Form using the **MdiChildBorder** property in the **ToolStripMenuItem_Click** event for the **New** button.

C# copyCode

```
private void newToolStripMenuItem_Click(object sender, EventArgs e)
{
    NewForm cfrm1 = new NewForm();
    //子C1RibbonFormのMdiChildBorderプロパティを設定します
    cfrm1.MdiChildBorder = 2;
    cfrm1.MdiParent = this;
    cfrm1.StartPosition = FormStartPosition.CenterScreen;
    cfrm1.Show();
}
```

5. Add the following code to create and display the MDI child form 'OpenForm' using the **ToolStripMenuItem_Click** event for the **Open** button.

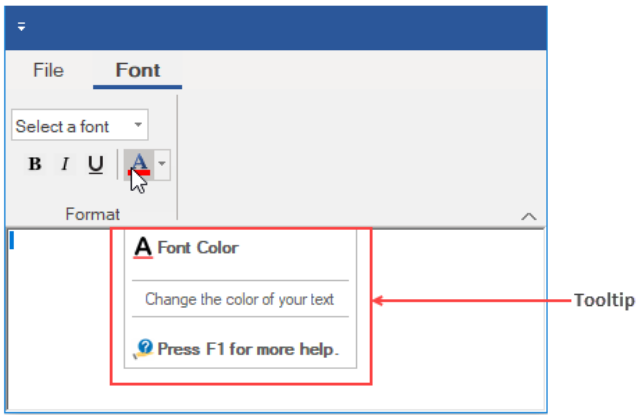
C# copyCode

```
private void openToolStripMenuItem_Click(object sender, EventArgs e)
{
    OpenForm cfrm2 = new OpenForm();
    cfrm2.Show();
    cfrm2.MdiParent = this;
}
```

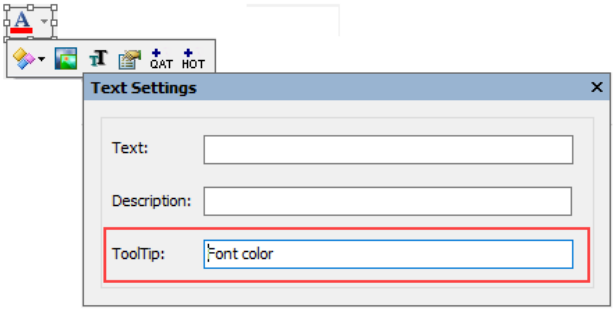
ツールチップ

A Tooltip is a pop-up window with string that appears when you hold a cursor over an item in an application. The information in the window is usually a very brief description about the item's purpose.

Every element in the **C1Ribbon** control has a **ToolTip** property, be it Ribbon Tab, Group, Group Items, QAT or Configuration Toolbar. The ribbon application below shows a tooltip on a ColorPicker when the cursor rests on it.



Tooltips can be added via the Text Menu of the floating toolbar of ribbon items.



The tooltip for an item can also be added via the Properties window.

A user can also add a tooltip to a Ribbon Item programmatically using the **TooTip ('ToolTip プロパティ' in the on-line documentation)** property of the **RibbonItem ('RibbonItem クラス' in the on-line documentation)** class.

- **Visual Basic**

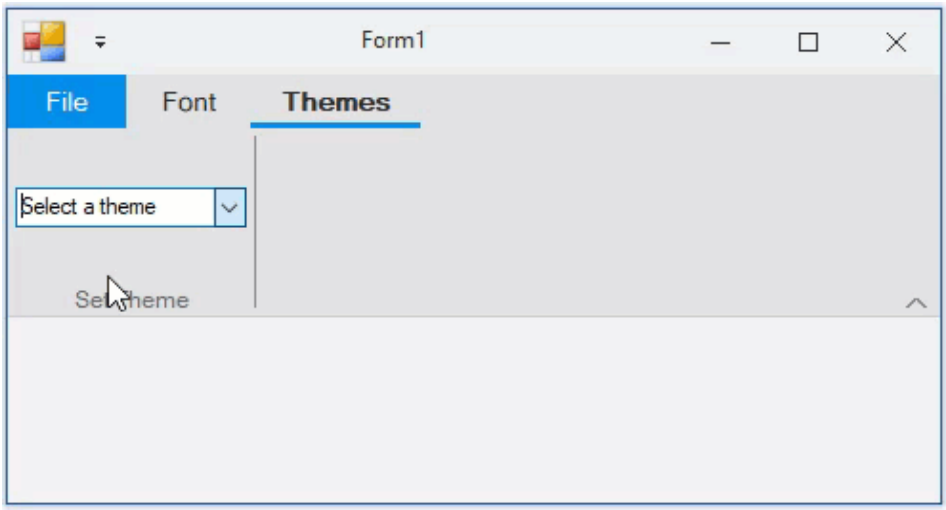
```
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        'FontComboBoxに対してツールチップを追加します
        fontComboBox.ToolTip = "Pick a new font for your text"
        '太字ボタンに対してリッチなツールチップを追加します
        boldButton.ToolTip = "<p><b>Bold (Ctrl + B)<b><p>" & "<p>Make your text Bold.</p> "
        '斜体ボタンに対してリッチなツールチップを追加します
        italicButton.ToolTip = "<p><b>Italic (Ctrl + I)<b><p>" & "<p>Italicize your text.</p> "
        '下線ボタンに対してリッチなツールチップを追加します
        underlineButton.ToolTip = "<p><b>Underline (Ctrl + U)<b><p>" & "<p>Underline your text.</p>"
        'HTMLコードを使用してカラーピッカーに対してリッチなツールチップを追加します(Officeタブ)
        colorPicker.ToolTip = "<table><tr><td><img src = 'fontcolor1.png'></td><td><b> Font Color </b></td></tr></table>"
& "<hr noshade size = 1 style = 'margin:2' color =#bebebe><div style = 'margin:1 12'>Change the color of your text</div>"
& "<hr noshade size = 1 style = 'margin:2' color =#bebebe><table><tr><td><img src = 'help1.png'></td>" & "<td><b> Press F1
for more help.</b></td></tr></table>"
    End Sub
```

- **C#**

```
// FontComboBoxに対してツールチップを追加します
fontComboBox.ToolTip = "Pick a new font for your text";

// HTMLを使用して太字、斜体、下線ボタンに対してリッチなツールチップを追加します
boldToggleButton.ToolTip = "<p><b>Bold (Ctrl + B)<b><p>" +
    "<p>Make your text Bold.</p> ";
italicToggleButton.ToolTip = "<p><b>Italic (Ctrl + I)<b><p>" +
    "<p>Italicize your text.</p> ";
underlineToggleButton.ToolTip = "<p><b>Underline (Ctrl + U)<b><p>" +
    "<p>Underline your text.</p>";

// HTMLコードを使用してカラーピッカーに対してリッチなツールチップを追加します
// (ツールチップエディターの[Office]タブで生成します)
colorPicker.ToolTip = "<table><tr><td><img src = 'fontcolor1.png'></td><td><b> Font Color </b></td></tr></table>" +
    "<hr noshade size = 1 style = 'margin:2' color =#bebebe><div style = 'margin:1 12'>Change the color of your text</div>" +
    "<hr noshade size = 1 style = 'margin:2' color =#bebebe><table><tr><td><img src = 'help1.png'></td>" +
    "<td><b> Press F1 for more help.</b></td></tr></table>";
```

テーマ

A user can apply themes to the C1Ribbon control in the WinForms application by selecting one of the built-in themes, which can be accessed by adding the reference assembly **C1.Win.C1Themes.4.5.2.**



The assembly provides the following built-in themes:

- BeigeOne
- ExpressionDark
- ExpressionLight
- GreenHouse
- MacBlue
- MacSilver
- Office2007Black
- Office2007Blue
- Office2007Silver
- Office2010Barbie
- Office2010Black
- Office2010Blue
- Office2010Green
- Office2010Red
- Office2010Silver
- Office2013DarkGray
- Office2013Gray
- Office2013Green
- Office2013HighContrast
- Office2013LightGray
- Office2013Red

- Office2013White
- Office2016Black
- Office2016Colorful
- Office2016DarkGray
- Office2016White
- RanierOrange
- ShinyBlue
- Violette
- VS2013Blue
- VS2013Dark
- VS2013DarkSolar
- VS2013Green
- VS2013Light
- VS2013Purple
- VS2013Red
- VS2013Tan
- Windows8Blue
- Windows8Brown
- Windows8Gray
- Windows8Green
- Windows8Red
- Material
- MaterialDark
- Office2016Green

Themes can be added to the **C1Ribbon** control only through the code. In order to add the Themes functionality to C1Ribbon, you can create a ComboBox using the ControlHost RibbonGroup Item and populate the ComboBox with the available themes. The user can then call the GetThemeByName method of the C1ThemeController class to retrieve the themes registered with the application by their names. Further, you can call the ApplyThemeToControlTree method to recursively apply a theme to a control and its children. This is depicted in the code snippet below:

- **Visual Basic**

```
Public Class ComboBoxHost
    Inherits RibbonControlHost
    Public Sub New()
        MyBase.New(New System.Windows.Forms.ComboBox())
        Dim ribbonBox = DirectCast(MyBase.Control, ComboBox)
        ribbonBox.Items.AddRange(C1ThemeController.GetThemes())
        ribbonBox.Text = "Select a theme"
        AddHandler ribbonBox.SelectedIndexChanged, AddressOf RibbonBox_SelectedIndexChanged
    End Sub

    Private Sub RibbonBox_SelectedIndexChanged(sender As Object, e As EventArgs)
        Dim themmeCombo As ComboBox = DirectCast(sender, ComboBox)
        Dim theme As C1Theme = C1ThemeController.GetThemeByName(themmeCombo.Text, False)
        C1ThemeController.ApplyThemeToControlTree(Me.Ribbon, theme)
    End Sub
End Class
```

- **C#**

```
    private void c1Ribbon1_RibbonEvent(object sender, RibbonEventArgs e)
    {
        //利用可能なテーマをコンボボックスに適用します
        var themeCombo = (ComboBox)comboBoxHost1.Control;
        themeCombo.Items.AddRange(C1ThemeController.GetThemes());
        themeCombo.Text = "Select a theme";
        themeCombo.SelectedIndexChanged += ThemeCombo_SelectedIndexChanged;
    }

    //選択したテーマをリボンに適用します
    private void ThemeCombo_SelectedIndexChanged(object sender, EventArgs e)
```

```
        {
            ComboBox themmeCombo = ((ComboBox)sender);
            C1Theme theme = C1ThemeController.GetThemeByName(themmeCombo.Text, false);
            C1ThemeController.ApplyThemeToControlTree(this.Ribbon, theme);
        }
}

//ControlHostのRibbonGroupアイテムを使用してComboBoxをリボンに追加します
public class ComboBoxHost : RibbonControlHost
{
    public ComboBoxHost() : base(new System.Windows.Forms.ComboBox())
    {

    }
}
```

アイコン

Icons are the visual representation of commands in a Ribbon, and can help the user easily navigate the application UI. The **Ribbon for WinForms** uses icons to display the different elements on the ribbon user interface. **C1Ribbon** uses C1Icon class, which is a part of C1.Framework namespace (C1.Win.4 and C1.Win.4.5 assemblies). **C1Icon** is a series of classes which allows to specify monochromatic icons that can be tinted and resized. These icons can be used internally in the controls. However, it also allows the customers to specify different icons through the API of the controls. The possible sources for creating icons can be Fonts, Vectors (Polygons or SVG) and Images. C1.Framework supports Bitmap, Vector, Font, Path, Polygon and Composite icon types. All the Ribbon elements support the **IconSet ('IconSet プロパティ in the on-line documentation)** property to define collection of images that should be used for different item size or DPI options.

The GIF below shows the Path, Font, Bitmap, Composite Icons and pressed Iconsets in the Ribbon control.

The different icons supported by C1Icon is listed in the table below:

Icon Type	Description
Bitmap Icon	BitmapIcon allows to wrap bitmap images from different sources.
Font Icon	FontIcon allows to use symbols from different fonts instead of images. Note that the Font icon cannot be resized automatically.
Path Icon	PathIcon allows to draw SVG paths (or XAML paths which have the same syntax). It rasterizes vector paths to bitmaps according to desired size and show bitmaps in UI.
Polygon Icon	PolygonIcon allows to draw polygons defined by points.
Composite Icon	Composite icons allow users to combine several icons into one. For example, the Ribbon Color Picker has a composite icon by default.
Vector Icon	Vector icons allows to represent an icon created from a vectorial definition.

In **C1Ribbon** control, all Ribbon elements use the **IconSet** property to define set of icons of different sizes to use in a single UI element. For each element which supports images, the application can define one or more icons. For example, the application can define 16x16 icon for small view, 20x20 icon for medium view, 32x32 icon for large view and (optionally) icons of bigger size for HighDPI environments. At runtime, **C1Ribbon** will automatically select the most appropriate icon for the current item type and screen resolution. If there is no predefined icon of such size, then **C1Ribbon** will resize the most appropriate icon from the specified icon set.

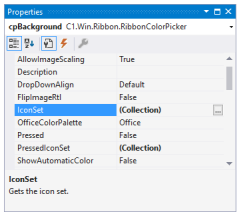
The Icons can be set in the Ribbon items both via the designer as well as code.

Implementation at Design-Time

For every element in C1Ribbon, you can add an icon with the help of **Select Image** editor. This editor allows to add and edit **C1BimtapIcon** , **C1FontIcon** and **C1PathIcon**.

The Select Image editor opens when a user clicks one of the four options in the **Change Image** button in the floating toolbar of a ribbon item.

The user can also evoke the **Select Image** editor by clicking the **IconSet** property of an item in the **Properties** window.



The **Select Image** editor contains three panels. The first panel contains a menu of buttons for Bitmap, Path and Font icons. The second panel contains the **Current Icon** properties, while the third panel contains the **Icons preset**, where the user can choose different icons and specify different preset sizes (16x16 pixels, 20x20 pixels and 32x32 pixels). Clicking each button displays the Current Icon properties and Icon preset for that button. For example, clicking the Bitmap icon displays the Current Icon properties and Icon preset options for bitmap icons. With the Current Icon properties list, the user can change the color, size, stroke, font, etc. of an icon. Further, there is an **Import** button to import images for the icon. Once you have selected a suitable Bitmap, Font or Path icon and modified its properties, you can click the OK button to confirm adding the icon to the ribbon item.Note that design-time support via the Select Image editor isn't available for Vector and Composite Icons as of now.

Implementation through Code

You can also add icons to elements in the Ribbon control using the C1BitmapIcon, C1FontIcon, C1VectorIcon, C1PathIcon and C1CompositeIcon classes.

To add a Bitmap icon, refer the code snippet below:

- **Visual Basic**

```
Public Sub AddBitapIcon()
    'Clear the IconSet Collection before adding new IconSet
    printButton.IconSet.Clear()
    'Add Icon Sets for Print Button
    'Large view
    printButton.IconSet.Add(New C1BitmapIcon("P", New Size(32, 32), Color.Transparent, Image.FromFile("images\print32.png")))
    'Medium View
    printButton.IconSet.Add(New C1BitmapIcon("P", New Size(20, 20), Color.Transparent, Image.FromFile("images\print20.png")))
    'Small View
    printButton.IconSet.Add(New C1BitmapIcon("P", New Size(16, 16), Color.Transparent, Image.FromFile("images\print16.png")))
End Sub
```

- **C#**

```
public void AddBitmapIcon()
{
    //Clear the IconSet Collection before adding new IconSet
    printButton.IconSet.Clear();

    //Add Icons of different sizes to the Print Button IconSet using C1BitmapIcon
    //These icons will be used as Button images depending on the size of the Ribbon
    printButton.IconSet.Add(new C1BitmapIcon("P", new Size(32, 32), Color.Transparent, Image.FromFile(@"images\print32.png")));
    printButton.IconSet.Add(new C1BitmapIcon("P", new Size(20, 20), Color.Transparent, Image.FromFile(@"images\print20.png")));
    printButton.IconSet.Add(new C1BitmapIcon("P", new Size(16, 16), Color.Transparent, Image.FromFile(@"images\print16.png")));
}
```

To add a Path icon, refer the code snippet below:

- **Visual Basic**

```
Public Sub AddPathIcon()
    'Add C1PathIcon to Button (Supports Scalable Vector Graphics)
    triangleShape.Text = "Triangle"
    circleShape.Text = "Circle"
    triangleShape.IconSet.Add(New C1PathIcon("T", New Size(32, 32), Color.Black, "M150 0 L75 200 L225 200 z"))
    circleShape.IconSet.Add(New C1PathIcon("C", New Size(32, 32), Color.Black, "M24.5,12.5 C24.5,19.127417 19.127417,24.5 12.5,24.5 C5.872583,24.5 0.5,19.127417 0.5,12.5 C0.5,5.872583 5.872583,0.5 12.5,0.5 C19.127417,0.5 24.5,5.872583 24.5,12.5 z"))
End Sub
```

- **C#**

```
public void AddPathIcon()
{
    //Apply a vector icon to button image using C1PathIcon
    triangleButton.IconSet.Add(new C1PathIcon("T", new Size(32,32), Color.Black, "M150 0 L75 200 L225 200 z"));
    circleButton.IconSet.Add(new C1PathIcon("C", new Size(32, 32), Color.Black, "M24.5,12.5 C24.5,19.127417 19.127417,24.5 12.5,24.5 C5.872583,24.5 0.5,19.127417 0.5,12.5 C0.5,5.872583 5.872583,0.5 12.5,0.5 C19.127417,0.5 24.5,5.872583 24.5,12.5 z"));
}
```

To add a Font icon, refer the code snippet below:

- **Visual Basic**

```
Public Sub AddFontIcon()
    'Add FontIcon for Bold Button
    Dim fontIconBold As C1FontIcon = New C1FontIcon()
    fontIconBold.Size = New Size(20, 20)
    fontIconBold.Text = "A"
    fontIconBold.Font = New Font("Times New Roman", 16.0F, FontStyle.Bold)
    'Add C1FontIcon
    textEffects.IconSet.Add(fontIconBold)
End Sub
```

- **C#**

```
public void AddFontIcon()
{
    //Configure RibbonSplitButton
    textEffectsButton.TextImageRelation = C1.Win.Ribbon.TextImageRelation.ImageBeforeText;
```

```
    //Clear the IconSet collection before adding new IconSet
    textEffectsButton.IconSet.Clear();

    //Apply image to textEffects RibbonSplitButton using C1FontIcon
    C1FontIcon fontIcon = new C1FontIcon();
    fontIcon.Size = new Size(20, 20);
    fontIcon.Text = "A";
    fontIcon.Font = new Font("Times New Roman", 16.0f, FontStyle.Bold);
    textEffectsButton.IconSet.Add(fontIcon);
}
```

To add a Composite Icon, refer the code snippet below:

- **Visual Basic**

```
Private Sub AddCompositeIcon()
    Dim ci = New C1CompositeIcon()
    ci.Size = New Size(16, 16)
    Dim bmp = New C1BitmapIcon With {
        .Size = New Size(16, 16),
        .Source = Image.FromFile("images\FillTool.png")
    }
    ci.Icons.Add(bmp)
    Dim pl = New C1PolygonIcon With {
        .AllowSmoothing = False,
        .Size = New Size(16, 16),
        .IsClosed = True,
        .Points = New PointF() {New PointF(0, 12), New PointF(15, 12), New PointF(15, 15), New PointF(0, 15)}
    }
    ci.Icons.Add(pl)
    compositeFill.IconSet.Clear()
    compositeFill.IconSet.Add(ci)
End Sub
```

- **C#**

```
private void AddCompositeIcon()
{
    // BackColor bmp + polygon
    var ci = new C1CompositeIcon();
    ci.Size = new Size(16, 16);

    // first image
    var bmp = new C1BitmapIcon
    {
        Size = new Size(16, 16),
        Source = Image.FromFile(@"images\FillTool.png")
    };
    ci.Icons.Add(bmp);

    // second image
    var pl = new C1PolygonIcon
    {
        AllowSmoothing = false,
        Size = new Size(16, 16),
        IsClosed = true,
        Points = new PointF[] { new PointF(0, 12), new PointF(15, 12), new PointF(15, 15), new PointF(0, 15) }
    };
    ci.Icons.Add(pl);

    //Clear the IconSet Collection before adding new IconSet
    compositeFill.IconSet.Clear();

    //Add composite icon to fill button
    compositeFill.IconSet.Add(ci);
}
```

To add iconsets for the pressed state, refer the following code:

- **Visual Basic**

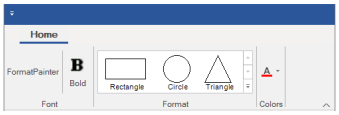
```
Public Sub CreatePressedIconSet()
        'Icon Set for Left Align Toggle Button
        leftAlign.IconSet.Add(New C1BitmapIcon("B", New Size(32, 32), Color.Transparent, Image.FromFile("images\align_left.png")))
        'Pressed Icon Set for Left Align Toggle Button
        leftAlign.PressedIconSet.Add(New C1BitmapIcon("Bold-Pressed", New Size(32, 32), Color.Transparent, Image.FromFile("images\align_left_pressed.png")))
        'Icon Set for Right Align Toggle Button
        rightAlign.IconSet.Add(New C1BitmapIcon("B", New Size(32, 32), Color.Transparent, Image.FromFile("images\align-right.png")))
        'Pressed Icon Set for Right Align Toggle Button
        rightAlign.PressedIconSet.Add(New C1BitmapIcon("Bold-Pressed", New Size(32, 32), Color.Transparent, Image.FromFile("images\align_right_pressed.png")))
    End Sub
```

- **C#**

```
public void CreatePressedIconSet()
{
    //IconSet for Left align Toggle Button
    leftAlign.IconSet.Add(new C1BitmapIcon("LA", new Size(32, 32), Color.Transparent, Image.FromFile(@"images\align_left.png")));

    //PressedIconSet for Left align Toggle Button
    leftAlign.PressedIconSet.Add(new C1BitmapIcon("LA", new Size(32, 32), Color.Transparent, Image.FromFile(@"images\align_left_pressed.png")));

    //IconSet for Right align Toggle Button
    rightAlign.IconSet.Add(new C1BitmapIcon("RA", new Size(32, 32), Color.Transparent, Image.FromFile(@"images\align-right.png")));

    //PressedIconSet for Right align Toggle Button
    rightAlign.PressedIconSet.Add(new C1BitmapIcon("RA", new Size(32, 32), Color.Transparent, Image.FromFile(@"images\align_right_pressed.png")));
}
```

Get Image from Icons

The Ribbon control lets you save the custom icon as image using the **GetItemImage** method of **C1Ribbon** class.



The **GetItemImage(C1.Win.Ribbon.RibbonItem item)** method gets an image rendered from the actual icon of the ribbon component. This method only works for visible ribbon.

To use the GetItemImage method, refer the following code:

C# copyCode

```
// Add a group named "Format" to the "Home" tab
RibbonGroup colorGroup = new RibbonGroup();
colorGroup.Text = "Colors";
homeTab.Groups.Add(colorGroup);

//Define a custom icon using C1PathIcon and C1CompositeIcon
var pathIcon = new C1PathIcon
{
    AllowSmoothing = false,
    Data = "M-2.7755575615628892e-17,18.89795939167988 h24 v5.142857098579968 h-24 z",
    Size = new Size(16, 16),
    Stroke = Color.Empty,
    Color = Color.Red
};
var compositeIcon = new C1CompositeIcon();
compositeIcon.Icons.Add(new C1BitmapIcon("FontColor", new Size(16, 16), Color.Transparent, Image.FromFile(@"Images\FontColor.png")));
compositeIcon.Icons.Add(pathIcon);
compositeIcon.Size = new Size(16, 16);
RibbonColorPicker colorPicker = new RibbonColorPicker();
colorPicker.IconSet.Add(compositeIcon);
colorPicker.ToolTip = "Color Picker";
colorGroup.Items.Add(colorPicker);

//Save custom icon as image using either of the approaches:
//Approach 1: Get custom icon image using GetItemImage method of C1Ribbon
Image img1 = c1Ribbon1.GetItemImage(colorPicker);
img1.Save("colorPicker.png");
```

Another approach to get the custom icon image is using the **GetImage** method of **IconRenderer** class.

C# copyCode

```
//Approach 2: Get custom icon image using GetImage method of IconRenderer
var render = IconRenderer.CreateRenderer(colorPicker.IconSet[0]);
Image img2 = render.GetImage();
img2.Save("colorPicker_renderer.png");
```

キーボードのサポート

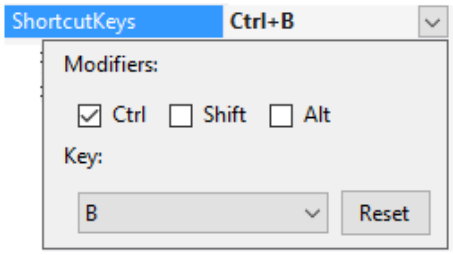
The **C1Ribbon** control supports keyboard shortcuts. You can perform tasks quickly by simply pressing a few keys. The user can access every command in the Ribbon using a key. For instance, to make the text bold, you can use the combination of **Ctrl** and **B** keys.

Note: This topic assumes that you have added a Ribbon button to the group and created a **RibbonButton.Click** event

> handler for the Bold button.

Complete the following steps for keyboard shortcuts.

1. Change the **Windows Form** to **Ribbon Form**.
2. Select the **Ribbon button** (in this case, the Bold button) to display its properties in the **Properties** window.
3. Locate the **ShortcutKeys** property and select the **Ctrl** check box and select **B** from the drop-down list.



4. Click outside the Shortcut Keys editor to accept the changes. Save and run the application.
   Now when you run the application, pressing the **CTRL+B** key combination will trigger the **RibbonButton.Click** event for the Bold button and make the selected text bold in style.

You can also add keyboard shortcuts to the Ribbon control programmatically using the **ShortcutKeyDisplayString ('ShortcutKeyDisplayString プロパティ' in the on-line documentation)** and **ShortcutKeys ('ShortcutKeys プロパティ' in the on-line documentation)** properties of **RibbonButton ('RibbonButton クラス' in the on-line documentation)** class.

- **Visual Basic**

```
Private Sub KeyboardShortcuts_Load(sender As Object, e As EventArgs) Handles MyBase.Load

    '太字ボタンのキーボードショートカットを設定します
    boldButton.ShortcutKeyDisplayString = "Bold"
    boldButton.ShortcutKeys = (Keys.Control Or Keys.B)

    '斜体ボタンのキーボードショートカットを設定します
    italicButton.ShortcutKeyDisplayString = "Italic"
    italicButton.ShortcutKeys = (Keys.Control Or Keys.I)

    '下線ボタンのキーボードショートカットを設定します
    underlineButton.ShortcutKeyDisplayString = "Underline"
    underlineButton.ShortcutKeys = (Keys.Control Or Keys.U)
End Sub


Private Sub boldButton_Click(sender As Object, e As EventArgs) Handles boldButton.Click
    RichTextBox1.SelectionFont = New Font("Times New Roman", 12, FontStyle.Bold)
End Sub

Private Sub italicButton_Click(sender As Object, e As EventArgs) Handles italicButton.Click
    RichTextBox1.SelectionFont = New Font("Times New Roman", 12, FontStyle.Italic)
End Sub

Private Sub underlineButton_Click(sender As Object, e As EventArgs) Handles underlineButton.Click
    RichTextBox1.SelectionFont = New Font("Times New Roman", 12, FontStyle.Underline)
End Sub
```

- **C#**

```
public KeyboardShortcuts()
{
    InitializeComponent();

    //太字ボタンのキーボードショートカットを設定します
    boldButton.ShortcutKeyDisplayString = "Bold";
    boldButton.ShortcutKeys = (Keys.Control | Keys.B);

    //斜体ボタンのキーボードショートカットを設定します
    italicButton.ShortcutKeyDisplayString = "Italic";
    italicButton.ShortcutKeys = (Keys.Control | Keys.I);
```

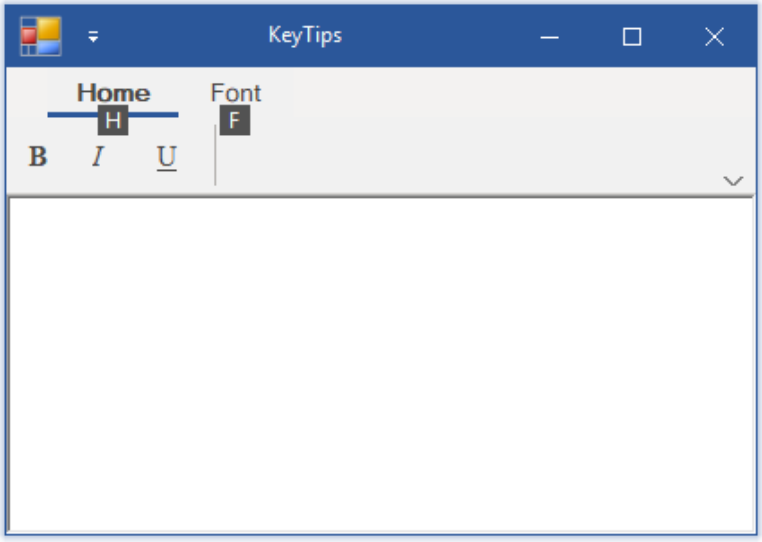
```
    //下線ボタンのキーボードショートカットを設定します
    underlineButton.ShortcutKeyDisplayString = "Underline";
    underlineButton.ShortcutKeys = (Keys.Control | Keys.U);

}

private void boldButton_Click(object sender, EventArgs e)
{
    richTextBox1.SelectionFont = new Font("Calibri", 12, FontStyle.Bold);
}

private void italicButton_Click(object sender, EventArgs e)
{
    richTextBox1.SelectionFont = new Font("Calibri", 12, FontStyle.Italic);
}

private void underlineButton_Click(object sender, EventArgs e)
{
    richTextBox1.SelectionFont = new Font("Calibri", 12, FontStyle.Underline);
}
```

Adding KeyTips to Ribbon Items

The **C1Ribbon** control also supports KeyTips, with which you can display all the commands on the Ribbon. It also allows you to move between elements on the C1Ribbon control using keyboard commands.

In order to see little boxes called Key Tips over each command, the user can press and release the **ALT**or **F10** key.

The image below illustrates the Key Tips for **Home** and **Font** tabs:



A user can make the key tips work at design time by setting the **KeyTip** property in the **Properties** Window. Every Item in the Ribbon control has a KeyTip property. For example, the user can add a Key Tip to an item using the **KeyTip ('KeyTip プロパティ' in the on-line documentation)** property of **RibbonButton ('RibbonButton クラス' in the on-line documentation)**. This is shown in the code snippet given below:

- **Visual Basic**

```
'ホーム・フォントタブのキーチップを設定します
homeTab.KeyTip = "H"
fontTab.KeyTip = "F"

'[ホーム]タブに太字・斜体・下線ボタンのキーチップを設定します
boldButton.KeyTip = "B"
italicButton.KeyTip = "I"
underlineButton.KeyTip = "U"

'[フォント]タブにFontComboBox・ColorPickerのキーチップを設定します
RibbonFontComboBox1.KeyTip = "T"
RibbonColorPicker1.KeyTip = "C"
```

- **C#**

```
//ホーム・フォントタブのキーチップを設定します
homeTab.KeyTip = "H";
fontTab.KeyTip = "F";

//[ホーム]タブに太字・斜体・下線ボタンのキーチップを設定します
boldButton.KeyTip = "B";
italicButton.KeyTip = "I";
underlineButton.KeyTip = "U";

//[フォント]タブにFontComboBox・ColorPickerのキーチップを設定します
ribbonFontComboBox1.KeyTip = "T";
ribbonColorPicker1.KeyTip = "C";
```

実行時のインタラクティブ操作

The Ribbon control can be customized at run-time according to your needs. For example, you can minimize the Ribbon to save space on the page and get a simplified view. You can also customize the Quick Access Toolbar (QAT) by adding or removing commands from it and moving it above or below the Ribbon.

This section will take you through the following:

Customizing the Quick Access Toolbar

Switching Ribbon Views

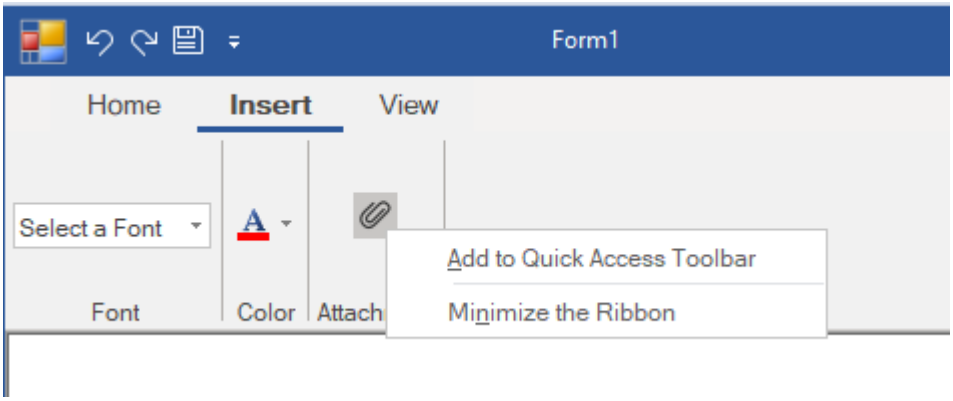
クイックアクセスツールバーのカスタマイズ化

The **Quick Access Toolbar (QAT)** can be customized in just a few click(s). You can add or remove items from the QAT as well as move the position of QAT above or below the Ribbon.

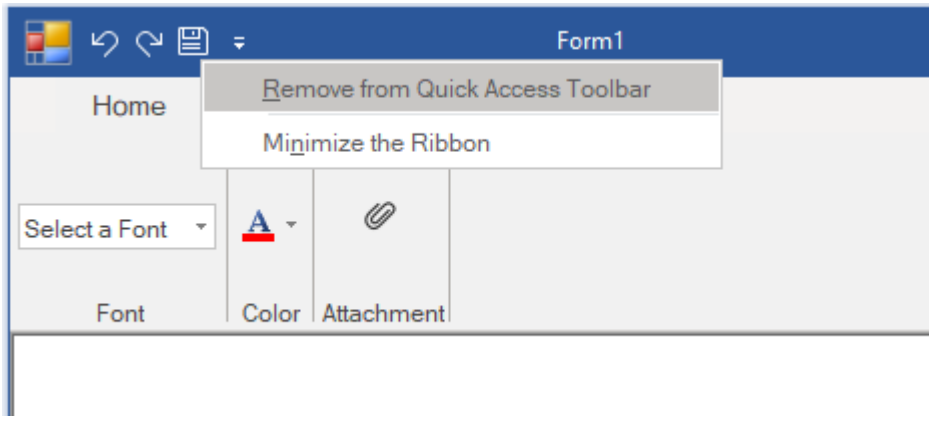
Add or Remove items from QAT

You can add the frequently-used commands from QAT easily.

To add a command to the QAT at run-time, simply right-click the item and select **Add to Quick Access Toolbar** from the context menu.



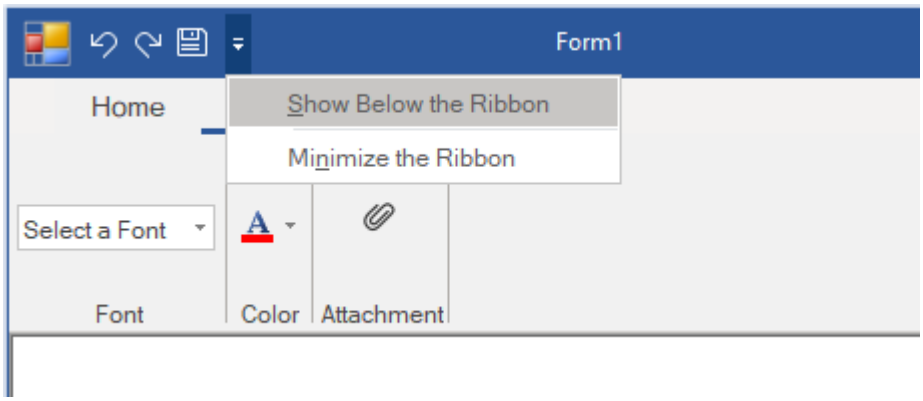
Similarly, you can remove any item from the QAT by right-clicking the item and selecting **RemovefromQuick Access Toolbar** from the context menu.



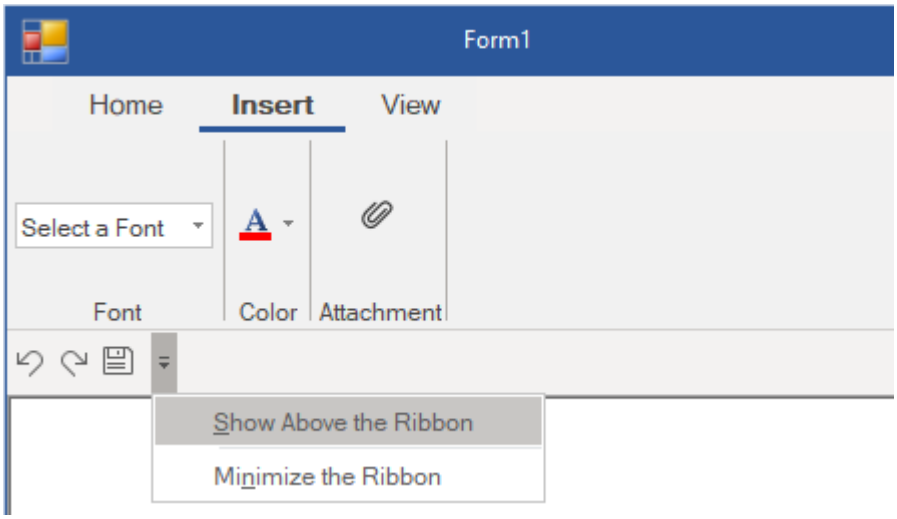
Move QAT

You can display the QAT above or below the Ribbon, depending upon your requirements.

To move the QAT below the Ribbon, click the drop-down arrow next to the QAT and select **Show Below the Ribbon**.



To move the QAT back to its location above the Ribbon, click the drop-down arrow next to the QAT and select **Show Above the Ribbon.**

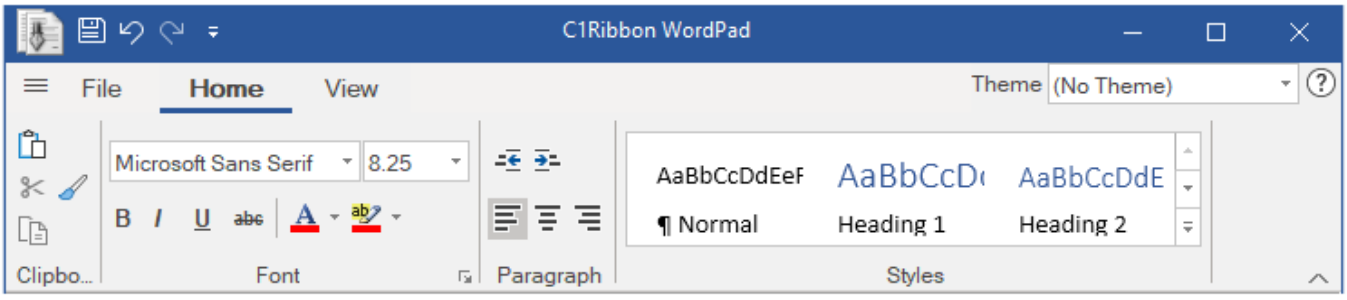


リボンビューの切り替え

The **C1Ribbon** control has three different view modes, **Full**, **Simplified** and **Minimized**. A user can switch from full view to simplified or minimized view. Switching ribbon views can increase the workspace area for the user.

Full View

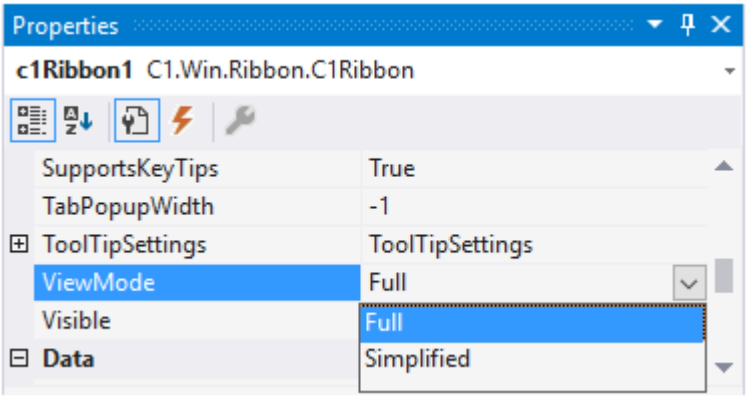
In **Full** or traditional view, the groups with commands are shown across three rows, giving the ribbon an expanded look. By default, the Ribbon control appears in full view mode.



Sometimes, there may not be enough room to display all the tabs and groups at the same level. In such cases, the Ribbon control supports smooth scrolling across tabs and groups by providing scroll buttons. This is very useful while resizing the ribbon control as well.

You can change the view of the Ribbon control by using the **ViewMode** property of **C1Ribbon** in the **Properties**

Window.



The user can programmatically change the ribbon view using the **ViewMode ('ViewMode プロパティ' in the on-line documentation)** property of **C1Ribbon ('C1Ribbon クラス' in the on-line documentation)** class. This is shown in the code below:

- **Visual Basic**

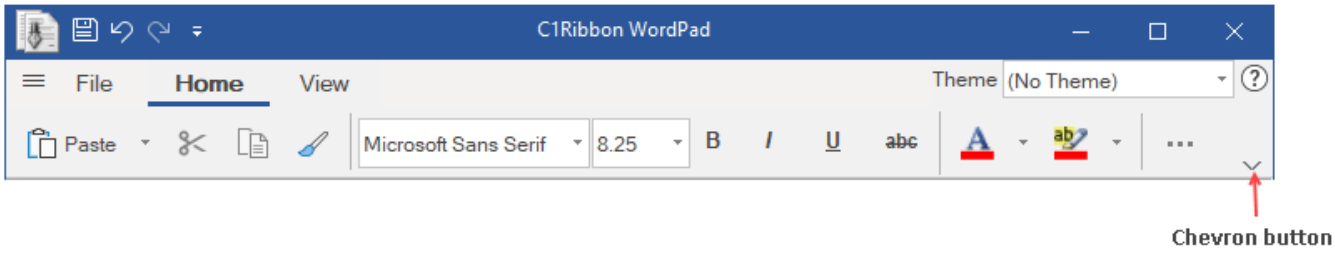
```
'Ribbon Full view
customRibbon.ViewMode = ViewMode.Full
'Ribbon Simplified view
customRibbon.ViewMode = ViewMode.Simplified
```

- **C#**

```
//Ribbon Full view
customRibbon.ViewMode = ViewMode.Full;
//Ribbon Simplified view
customRibbon.ViewMode = ViewMode.Simplified;
```

Simplified View

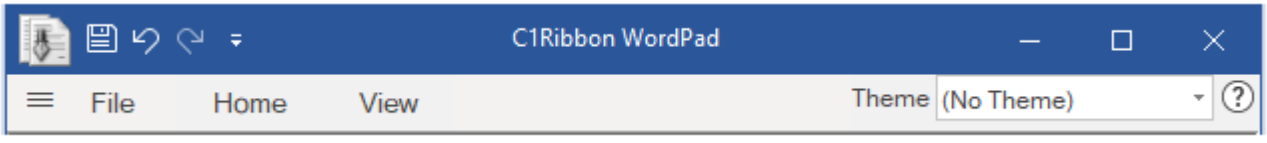
The **Simplified** view makes the ribbon compact by displaying all the commands in a single line without showing the groups. You can switch to the simplified view from full view by using the chevron button at the bottom right corner of the ribbon.



For more information about Simplified view, refer the Simplified Ribbon topic.

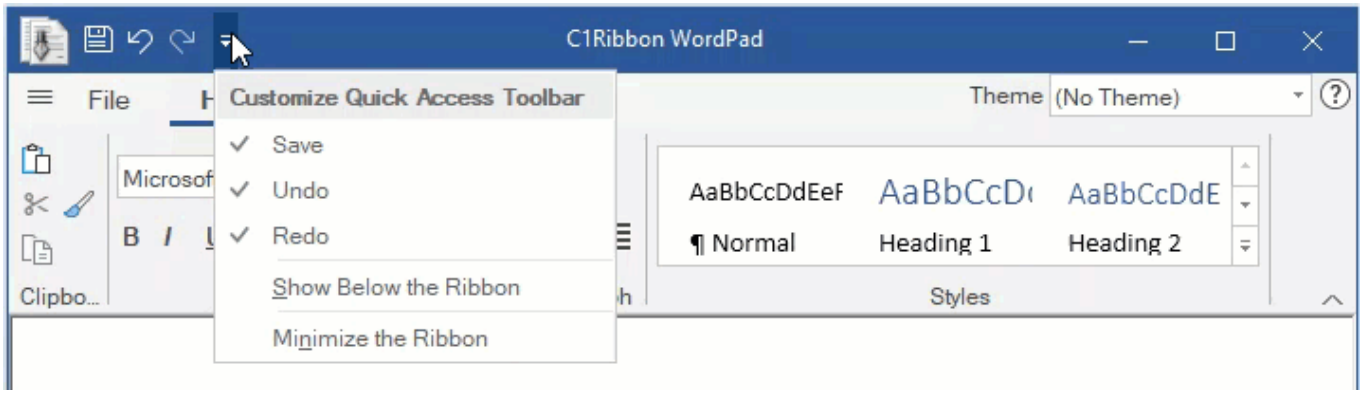
Minimized View

In **Minimized** view mode, a user can view only the tabs. The user can click on each tab to see the groups and items within that tab. The minimized view lets the ribbon save up more space than the simplified view mode.



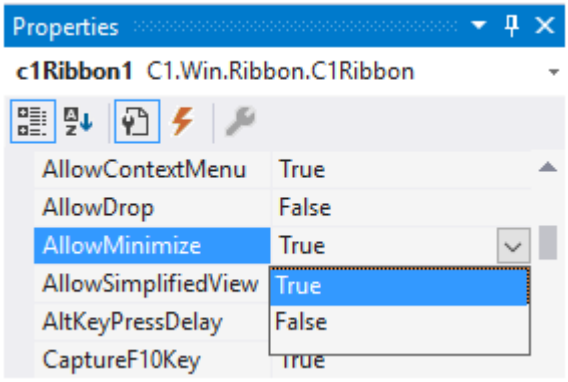
A user can switch to the minimized view using the QAT dropdown menu by selecting the **Minimize the Ribbon**

option. Further, you can also right click on the tabs, groups or items (buttons, combo-boxes, text-boxes etc.) and select the **Minimize the Ribbon** option to view the Ribbon control in its minimized state.



In order to get back to the earlier view from minimized view, you can click on the **Minimize the Ribbon** option from the QAT or tabs.

The Minimized view mode feature is already present in the C1ribbon control by default, but you can disable it at design-time using the **AllowMinimize** property of **C1Ribbon** in the properties window.



This can also be done via code using the **AllowMinimize ('AllowMinimize プロパティ' in the on-line documentation)** property of **C1Ribbon ('C1Ribbon クラス' in the on-line documentation)** class.

- **Visual Basic**

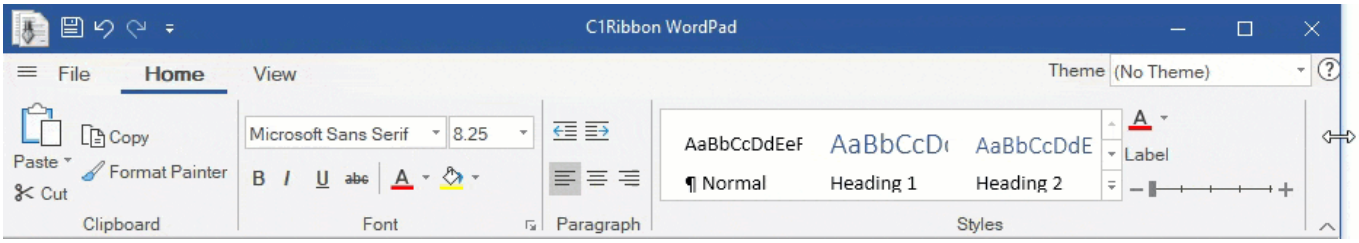
```
'Disable minimized view
customRibbon.AllowMinimize = False
```

- **C#**

```
//Disable minimized view
customRibbon.AllowMinimize = false;
```

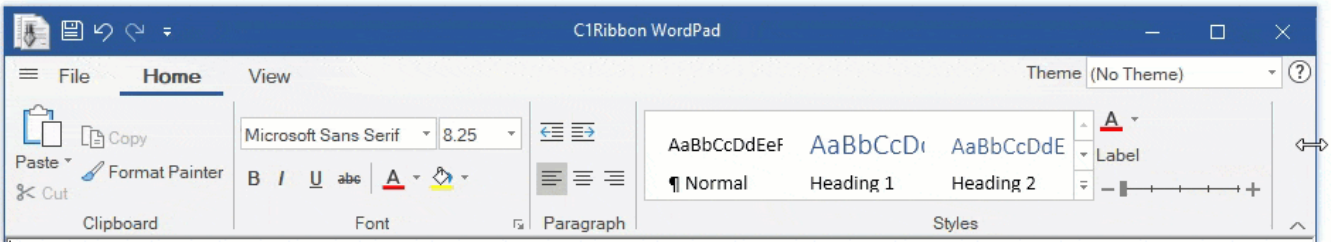
グループの折りたたみ

Ribbons are control-rich UIs that take up a good amount of screen real estate in comparison to menu applications. By default, the ribbon groups collapse left to right one by one individually into small rectangles. Also, in place of their constituent controls, these rectangles display a drop-down arrow, which when clicked shows the controls from the collapsed group in the dropdown menu.

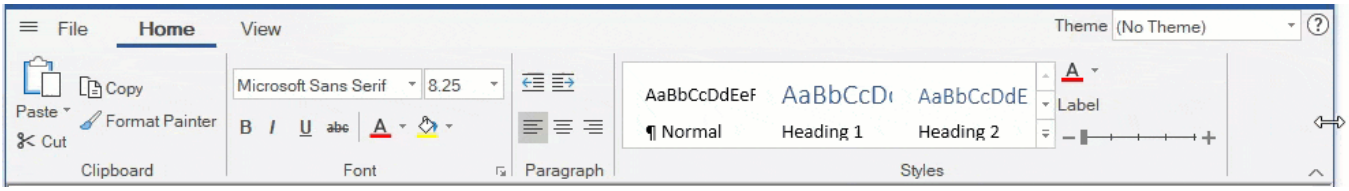


To let you efficiently use the space while horizontally resizing a form containing a Ribbon, the Ribbon control provides the **CollapseIndex ('CollapseIndex プロパティ' in the on-line documentation)** property in the **RibbonGroup ('RibbonGroup クラス' in the on-line documentation)** class. This property sets the priority index of the groups for resizing and collapsing. The higher the number, the lower the priority for resizing.

Let's say you have controls in the right half of the ribbon that are 'less important' than some groups in the left half and want the right group to collapse first in the application. For this, you have to set the **CollapseIndex** of the Styles and Paragraph groups to a value that is higher than the CollapseIndex numbers of Clipboard and Font groups.



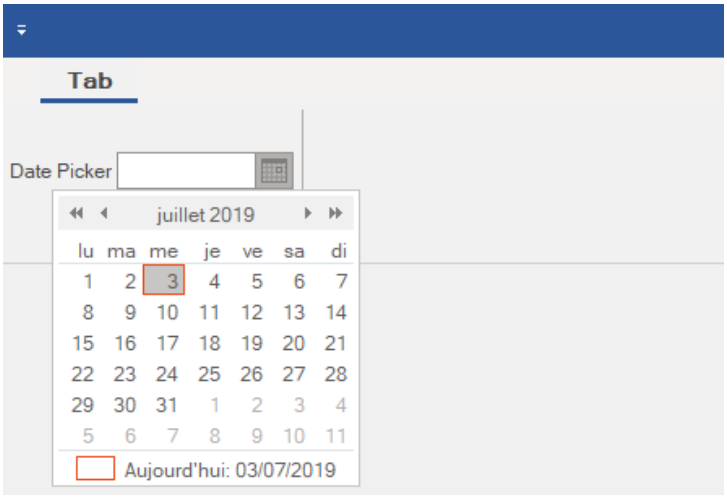
Likewise, if you want to initially collapse the Font and Paragraph groups while resizing the ribbon, you have to set their Collapse Indices to a values much higher than for the other groups, Clipboard and Styles.



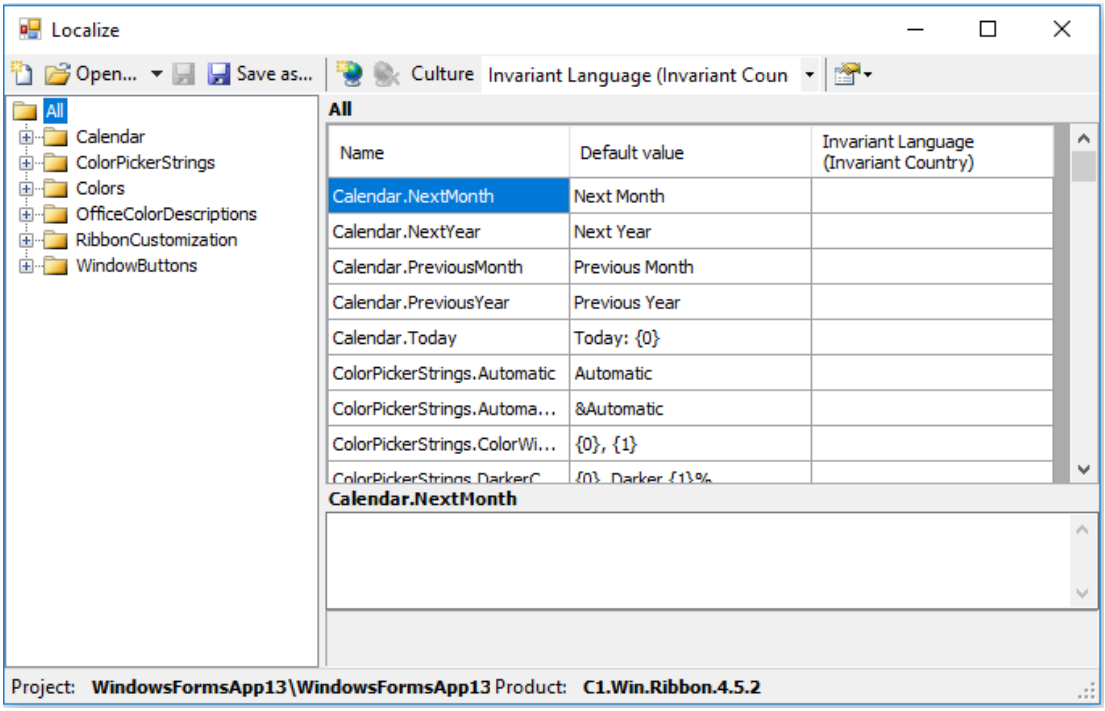
So, now you can easily collapse or resize groups as per the priority you want to set in the Ribbon control using the CollaspeIndex property.

ローカライゼーションの適用

The Ribbon control is embedded with a powerful localization feature which allows you to interact with it in different languages. The following image shows the Ribbon control when its localization is set to French.



Localize Dialog Box



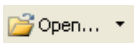
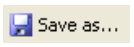
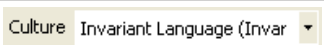
The left pane of the Localize dialog box displays a tree listing the localizable string IDs, and on the right are the strings themselves. The string list contains the following columns:

Column	Description
Name	The string's name (ID)
Default value	The default (English) value of the string.
Value	The string value for the currently selected culture (the column header displays that culture).

Below the list is the currently selected string's value, along with an optional description.

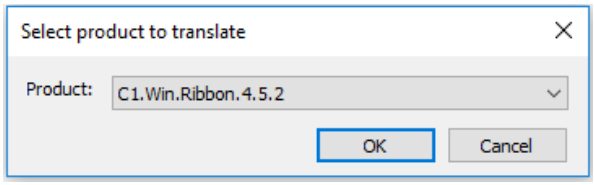
The status bar displays the project which will contain the localized resources and the name of the ComponentOne assembly which is currently being localized.

The Localize dialog box contains the following toolbar menu buttons:

Button	Description
	**Create new translation** begins a new localization for a ComponentOne assembly.
	**Open** opens an existing translation for a particular assembly.
	**Save** saves the current translation.
	**Save as** saves the current translation and allows you to select the project in which to save the translation.
	**Add culture** adds a new culture.
	**Delete culture** removes a culture from the translations.
	**Select culture** selects the culture to display and edit.
	**Options** customizes the appearance and behavior of the localization window.

Create New Translation

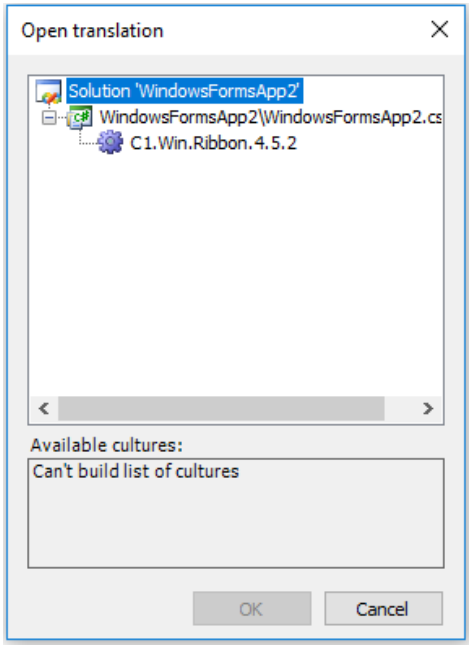
Clicking the **Create new translation** button begins a new localization for a ComponentOne assembly. A dialog box opens for you to select the ComponentOne assembly to localize.



Note: The assembly must be referenced in the currently open solution.

Open

Clicking the **Open** button opens an existing translation for a particular assembly. All translations that you create are stored as **.resx** files and are automatically added to the project that you select while saving the translation. Clicking this item shows a dialog box where a previously saved translation can be selected.



After you have created and saved a translation, the **Available cultures** panel shows the list of cultures for which translations were created for the selected assembly.

Save

Clicking the **Save** button saves the current translation.

The translation is saved in the project shown in the status bar. When the translation is saved, a folder with the name **C1LocalizedResources** is created in the selected project (if it does not already exist), and the .resx files with translations are saved in that folder and added to the project.

For each culture, a separate .resx file is created. These files are visible in the Solution Explorer window.

> **Note:** If your translation is only for the invariant culture, the .resx file does not contain a culture suffix.

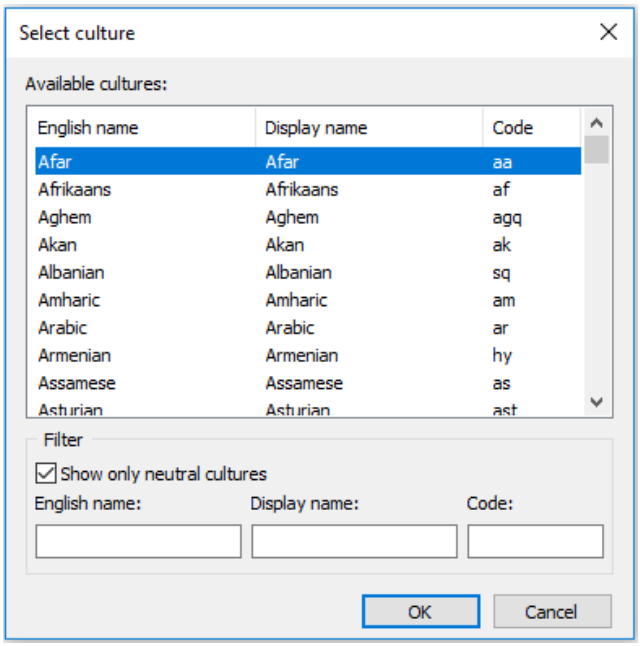
Save As

Clicking the **Save as** button saves the current translation and allows you to select the project in which to save the translation.

Add Culture

Clicking the **Add culture** button adds a new culture.

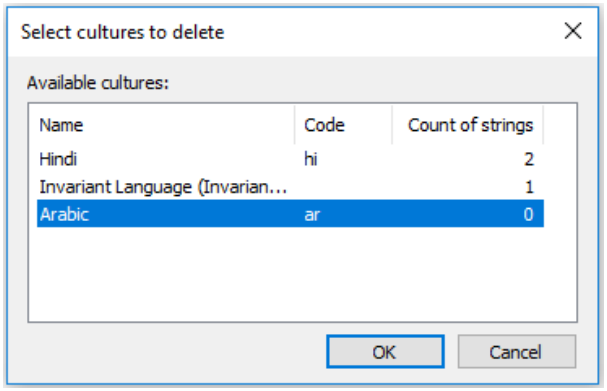
You can make translations for several cultures. For each culture, a separate .resx file is created in the **C1LocalizedResources** folder. Clicking the **Add culture** button opens the **Select culture** dialog box that provides a list of available cultures:



Initially the list contains neutral cultures only. To show all cultures, uncheck the **Show only neutral cultures** checkbox. You can use the **English name**, **Display name**, and **Code** boxes to filter the list of shown cultures. After you have selected a culture, press the **OK** button to add it to the translations. The newly added culture will appear in the **Culture** drop-down in the toolbar and will become current in the window.

Delete Culture

Clicking the **Delete culture** button removes a culture from the translations. The **Select cultures to delete** dialog box provides the list of cultures existing in the translations:



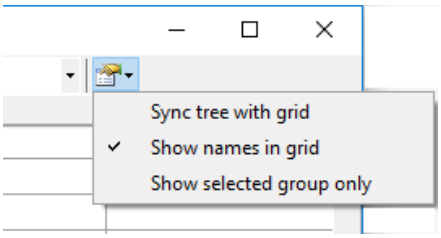
Selecting a culture and clicking **OK** removes it from the translations.

Select Culture

The **Culture** drop-down allows you to select the culture to display and edit.

Options

Clicking the **Options** button allows you to customize the appearance and behavior of the localization window.



The available localization options include:

Option	Description
Sync tree with grid	When this item is checked, selecting a string in the right panel list also selects that string in the tree on the left. By default this item is unchecked.
Show names in grid	When this item is checked, the *Name* column is shown in the right-hand panel, otherwise that column is hidden. By default this item is checked.
Show selected group only	When this item is checked, the list of strings on the right contains only the strings from the group currently selected in the tree on the left. By default this item is unchecked.

Setting the Current Culture

The Ribbon control uses the localization files automatically according to the culture selected in the application as long as the files have not been moved to another location or excluded from the project. The CurrentCulture property looks for the default strings of the current culture whereas the CurrentUICulture looks for the culture-specific resources at run-time. By default, the current culture is designated as System.Threading.Thread.CurrentThread.CurrentCulture. If you want to use a culture other than the current culture, you can set the desired culture in your application using the following code:

- **Visual Basic**

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    System.Threading.Thread.CurrentThread.CurrentUICulture = New System.Globalization.CultureInfo("fr-FR")
    System.Threading.Thread.CurrentThread.CurrentCulture = New System.Globalization.CultureInfo("fr-FR")
    InitializeComponent()
End Sub
```

- **C#**

```
public Form1()
{
    // 現在のカルチャをフランス語に設定します
    System.Threading.Thread.CurrentThread.CurrentUICulture = new System.Globalization.CultureInfo("fr-FR");
    System.Threading.Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo("fr-FR");

    InitializeComponent();
}
```

.NET 6 リファレンス

The following topics contain the API reference for **Ribbon for WinForms for .NET 5 assembly.**