ComponentOne

Windows 7 Control Pack for WinForms

ComponentOne, a division of GrapeCity
201 South Highland Avenue, Third Floor
Pittsburgh, PA 15206 USA

Website: http://www.componentone.com
Sales: sales@componentone.com
Telephone: 1.800.858.2739 or 1.412.681.4343 (Pittsburgh, PA USA Office)

Trademarks

Warranty

ComponentOne warrants that the media on which the software is delivered is free from defects in material and workmanship, assuming normal use, for a period of 90 days from the date of purchase. If a defect occurs during this time, you may return the defective media to ComponentOne, along with a dated proof of purchase, and ComponentOne will replace it at no charge. After 90 days, you can obtain a replacement for the defective media by sending it and a check for $2 5 (to cover postage and handling) to ComponentOne.

Except for the express warranty of the original media on which the software is delivered is set forth here, ComponentOne makes no other warranties, express or implied. Every attempt has been made to ensure that the information contained in this manual is correct as of the time it was written. ComponentOne is not responsible for any errors or omissions. ComponentOne's liability is limited to the amount you paid for the product. ComponentOne is not liable for any special, consequential, or other damages for any reason.

Copying and Distribution

Table of Contents

Windows 7 Control Pack for WinForms Overview

Integrate your applications with Windows 7 using **Windows 7 Control Pack for WinForms**. These controls enable you to easily manage progress indicators, thumbnails and jump lists on the Windows 7 taskbar, as well as create and display custom dialog boxes.

C1TaskbarButton works with Windows 7 taskbar extensions. You can adjust the application's Jump List, thumbnail toolbar, icon overlay, and a progress bar displayed on the taskbar button. Also, you can select a particular control on a form to use its contents as the thumbnail image.

C1TaskDialog is a powerful replacement for the standard message box under Windows Vista or newer operating systems. Similar to other standard dialog boxes (such as **OpenFileDialog**) you can set simply a few properties, then call to the Show() method to display the dialog box at run-time.

Getting Started

Get started with the following topics:

- Key Features
- Quick Start
- Components
- Samples

Help with WinForms Edition

Getting Started

For information on installing **ComponentOne Studio WinForms Edition**, licensing, technical support, namespaces and creating a project with the control, please visit Getting Started with WinForms Edition.

Key Features

Windows 7 Control Pack for WinForms incorporates several key features, including the following:

- **Manage Jump Lists**

  Jump lists display when you right-click your application's button on the Windows 7 taskbar. They allow quick access to recently used files or frequently used commands within your application. **C1TaskbarButton** enables you to manage your form's jump list, which can include lists of items or tasks.

- **Show Progress Indicators**

  With **C1TaskbarButton** you can easily display progress indicators within your application's taskbar button. Manage the progress value and choose a predefined state: Normal (Green), Error (Red), Paused (Yellow) or Indeterminate (Marquee).

- **Manage Thumbnail Buttons**

  Add and manage thumbnail buttons in the taskbar button flyout with **C1TaskbarButton**.

- **Display an Overlay Icon**

  With **C1TaskbarButton** you can show an overlay icon on the taskbar button. Use this feature to show status updates to the user even when the application is minimized.

- **Windows 7 and Vista-style Dialogs**

  Create and customize task dialogs without much effort. The **C1TaskDialog** component works like the common Windows dialogs (such as **OpenFileDialog**). You can set a few properties and then simply call the Show() method to display the dialog at run-time.

- **Customize Dialog Buttons**

  With **C1TaskDialog** you can completely customize the buttons on the dialog by just setting a few properties. Add custom buttons, radio buttons, check boxes, hyperlinks, expandable footer areas, a progress bar and more.

- **.NET 2.0 Support**

  Windows 7 Control Pack for WinForms can be used in .NET 2.0 and higher targeting applications. The client must have Windows 7 OS in order to see the features of **C1TaskbarButton** (Vista is the minimum requirement for **C1TaskDialog**). Clients who do not meet the requirement will simply not see the feature (no crash or exception).

Windows 7 Control Pack for WinForms Quick Starts

In this section you'll learn how to use the basic **Windows 7 Control Pack for WinForms** functionality to create a customized dialog box using the C1TaskDialog control. You'll also customize the taskbar using the C1TaskbarButton control. This section is not intended to be a comprehensive tutorial of all features of **Windows 7 Control Pack for WinForms**, but rather provide a quick introduction and highlight some general approaches to using the product.

C1TaskDialog Quick Start

In this section you'll learn how to use the basic **Windows 7 Control Pack for WinForms** functionality to create a customized dialog box using the C1TaskDialog control.

Step 1 of 4: Creating a C1TaskDialog Application

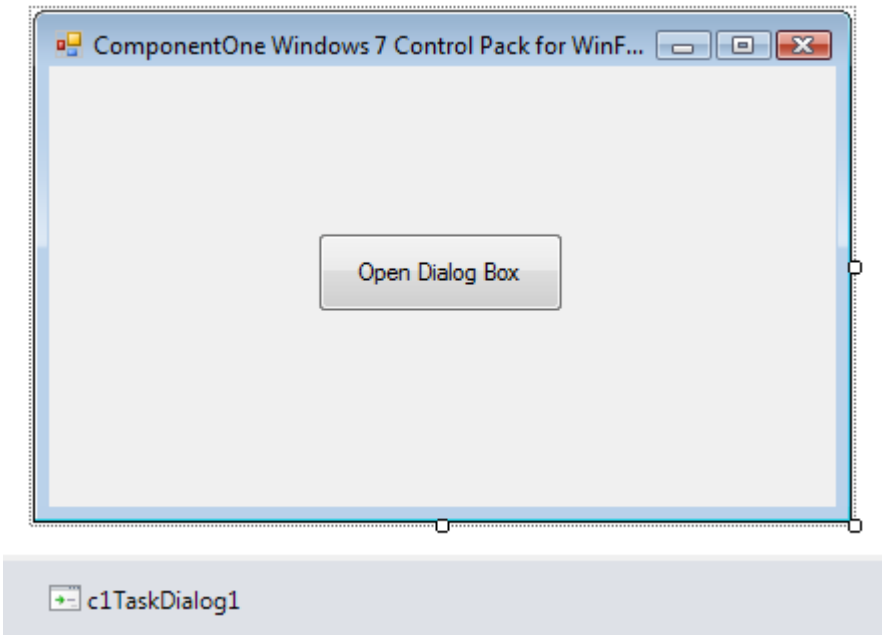
In this step you'll create a simple WinForms application and add the C1TaskDialog component to the application.

To begin, complete the following steps:

1. From the **File** menu in Microsoft Visual Studio, select **New** and click **Project**. The **New Project** dialog box opens.
2. Under **Project Types**, choose either **Visual Basic Projects** or **Visual C# Projects**, and select **Windows Application** from the list of **Templates** in the right pane.
3. Enter or browse for a location for your application in the **Location** field and click **OK**.
   A new WinForms project is created in the specified location. In addition, a new Form is displayed in the Designer view.
4. Resize the form in Design View.
5. Navigate to the Toolbox and double-click the C1TaskDialog item to add it to the application. The C1TaskDialog component will appear in the component tray below the form.
6. Click once on the form to select it then navigate to the Toolbox and double-click on the **Button** item to add a **Button** control to your application. Resize and position the **Button** control on the form.
7. Select the **Button** control and in the Properties window set its **Text** property to "Open Dialog Box".

What You've Accomplished

In this step you created a new application and added **Button** and **C1TaskDialog** controls to the application. The form should appear similar to the following:



In the next step you'll customize the appearance and behavior of the C1TaskDialog control.

Step 2 of 4: Customizing the C1TaskDialog Application

In the previous step you created a simple WinForms application and added the C1TaskDialog component to the application. In this step, you'll customize the appearance of the C1TaskDialog without adding any code to your project.

To begin, complete the following steps:

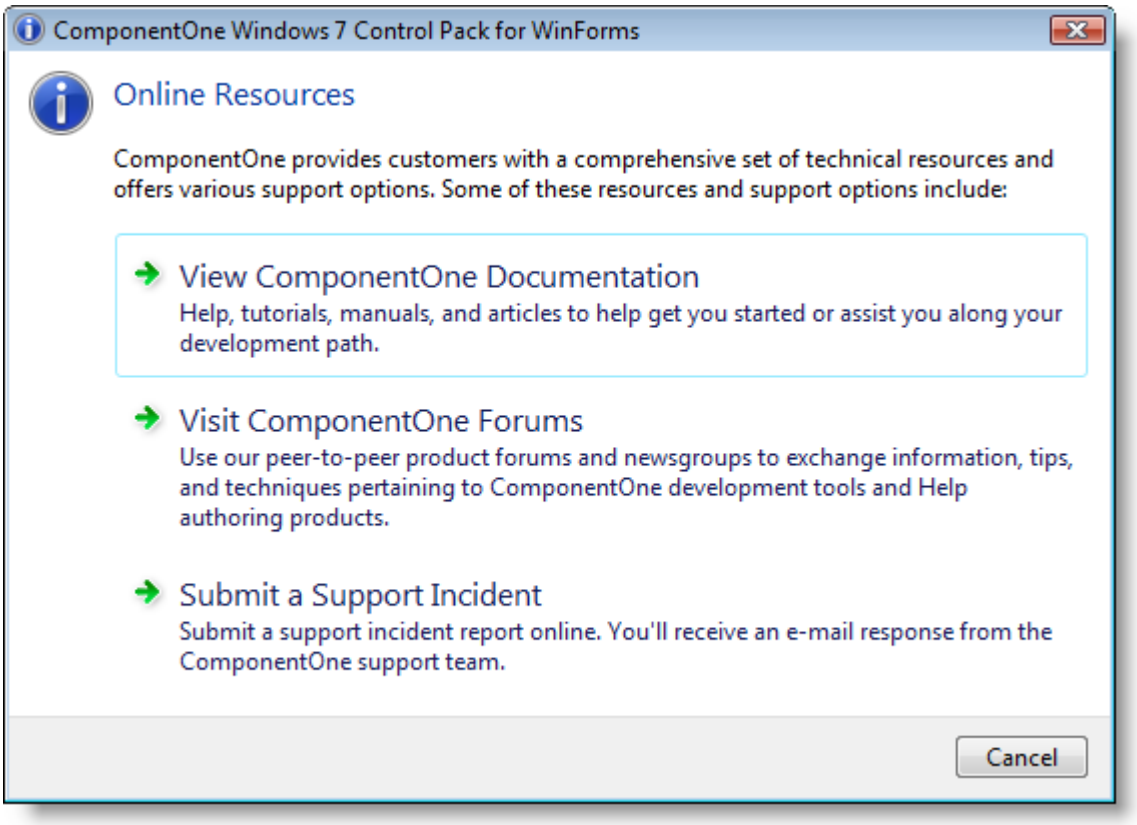
1. In Design view, click the C1TaskDialog once to select it, and locate and click the component's smart tag to open the **C1TaskDialog Tasks** menu.
2. Click the **Edit Custom Buttons** option to open the **C1TaskDialog.CustomButtons Collection Editor**. You'll add custom buttons in this dialog box.
3. In the **C1TaskDialog.CustomButtons Collection Editor**, click the **Add** button three times to add three custom buttons to your application.
4. Set the following options to customize the buttons:
    - Set **Button1**'s **Text** to "View ComponentOne Documentation" and **Note** item to "Help, tutorials, manuals, and articles to help get you started or assist you along your development path."
    - Set **Button2**'s **Text** to "Visit ComponentOne Forums" and **Note** item to "Use our peer-to-peer product forums and newsgroups to exchange information, tips, and techniques pertaining to ComponentOne development tools and Help authoring products."
    - Set **Button3**'s **Text** to "Submit a Support Incident" and **Note** item to " Submit a support incident report online. You'll receive an e-mail response from the ComponentOne support team."
5. In the **C1TaskDialog.CustomButtons Collection Editor**, click the OK button to close the dialog box and add the custom buttons you just created.
6. Click the C1TaskDialog once to select it, and locate and click the component's smart tag to open the **C1TaskDialog Tasks** menu.
7. Set the following properties to customize the appearance and behavior of the C1TaskDialog control:
    - Set the **Window Title** item to "ComponentOne Windows 7 Control Pack for WinForms".
    - Set the **Main Icon** item to **Information**.
    - Set the **Main Instructions** item to "Online Resources".
    - Set the **Content** item to "ComponentOne provides customers with a comprehensive set of technical resources and offers various support options. Some of these resources and support options include:".
    - Confirm that the **Use Command Links** check box is selected.

- Select the **Common Buttons** drop-down arrow and choose **Cancel** from the list of buttons.

8. Click elsewhere on the form to close the **C1TaskDialog Tasks** menu.

What You've Accomplished

In this step you customized the **C1TaskDialog** control's appearance and behavior. If you select the **Show Dialog** option from the **C1TaskDialog Tasks** menu, you'll see that the dialog box now looks similar to the following:



In the next step you'll add code to complete an action when the **Command Link** buttons are click and to open the dialog box when the button on the form is clicked at run time.

Step 3 of 4: Adding Code to the C1TaskDialog Application

In the last step you created a new application and added **Button** and **C1TaskDialog** controls. You also customized the appearance and behavior of the **C1TaskDialog** control. In this step you'll add code to complete an action when the **Command Link** buttons are click and to open the dialog box when the button on the form is clicked at run time.

Complete the following steps:

1. In Design view, double-click the **Button** control to switch open the Code Editor and create the **Button_Click** event handler.
2. Add code to the **Button_Click** event handler so that the dialog box will open when the button is clicked; the event handler will appear similar to the following:

To write code in Visual Basic

Visual Basic

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
    Me.C1TaskDialog1.Show()
End Sub
```

To write code in C#

C#

```
private void button1_Click(object sender, EventArgs e)
{
    this.c1TaskDialog1.Show();
}
```

3. Return to Design view and select the **C1TaskDialog** control.
4. Navigate to the Properties window and select the lightning bolt **Events** button to view the **C1TaskDialog** control's events.
5. Double-click the **ButtonClick** event to create a new event handler and return to Code view.
6. Add code to the **ButtonClick** event handler so that click each Command Link navigates to a Web site; the event handler will appear similar to the following:

To write code in Visual Basic

Visual Basic

```
Private Sub C1TaskDialog1_ButtonClick(ByVal sender As System.Object, ByVal e As
C1.Win.C1Win7Pack.TaskDialogButtonClickEventArgs) Handles C1TaskDialog1.ButtonClick
    If e.DialogResult = TaskDialogResult.[Custom] Then
        Select Case e.CustomButton.Name
            Case "Button1"

System.Diagnostics.Process.Start("http://www.componentone.com/SuperProducts/SupportServices/Documentation/")
                e.Cancel = True
                Exit Select
            Case "Button2"
                System.Diagnostics.Process.Start("http://www.componentone.com/forums/")
                e.Cancel = True
                Exit Select
            Case "Button3"
                System.Diagnostics.Process.Start("http://www.componentone.com/support/")
                Exit Select
                e.Cancel = True
        End Select
    End If
End Sub
```

To write code in C#

C#

```
private void c1TaskDialog1_ButtonClick(object sender, TaskDialogButtonClickEventArgs e)
{
    if (e.DialogResult == TaskDialogResult.Custom)
    {
        switch (e.CustomButton.Name)
        {
            case "Button1":

System.Diagnostics.Process.Start("http://www.componentone.com/SuperProducts/SupportServices/Documentation/");
                e.Cancel = true;
                break;
            case "Button2":
                System.Diagnostics.Process.Start("http://www.componentone.com/forums/");
                e.Cancel = true;
                break;
            case "Button3":
               System.Diagnostics.Process.Start("http://www.componentone.com/support/");
               break;
               e.Cancel = true;
        }
    }
}
```

What You've Accomplished

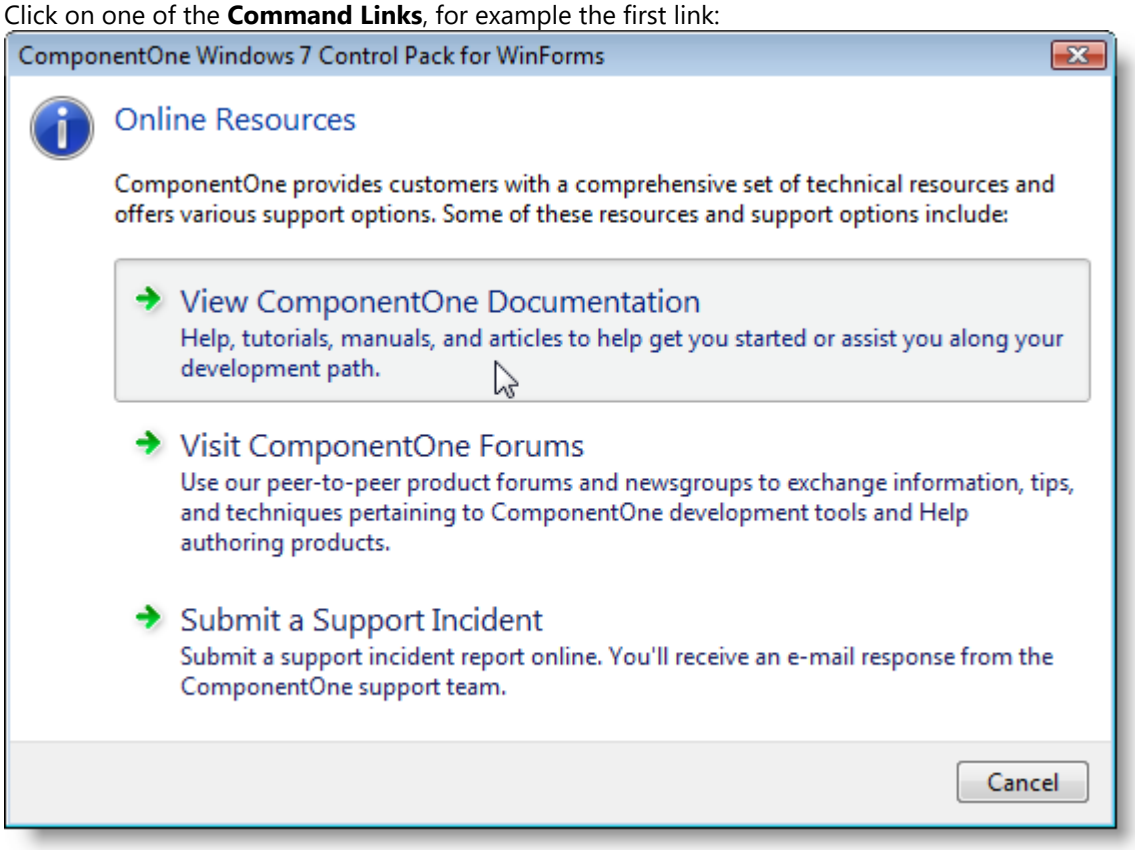
In this step you added code to your application to customized and initialized the **C1TaskDialog** control. In the next step you'll run the application to view some of the possible run-time interactions.

Step 4 of 4: Running the C1TaskDialog Application

In the previous steps of this quick start you created a new application, added the **C1TaskDialog** control, and customized the application. All that's left is to run the application to view some of the possible run-time interactions.

Complete the following steps to view some of the run-time interactions possible with your application:

1. From the **Debug** menu, select **Start Debugging** to view how your application will appear at run time.
2. Click the **Open Dialog Box** button that appears on the form. The **C1TaskDialog** dialog box will appear.
3. Click on one of the **Command Links**, for example the first link:



   The link will open in a browser window. Notice that the dialog box does not close when the link is clicked – this is because the code you added in the previous step cancelled the dialog box closing.

4. Click the **Cancel** button. The dialog box will close.

Congratulations, you have completed the **C1TaskDialog** quick start! You created a new application, added the **C1TaskDialog** control and customized it, added code to initialize the application, and run the application to view some of the possible run-time interactions.

C1TaskbarButton Quick Start

In this section, you'll learn how to use the basic **Windows 7 Control Pack for WinForms** functionality to create a taskbar button using the C1TaskbarButton control. You will also create and activate a jump list that contains jump links and jump tasks.

Step 1 of 4: Creating a C1TaskbarButton Application

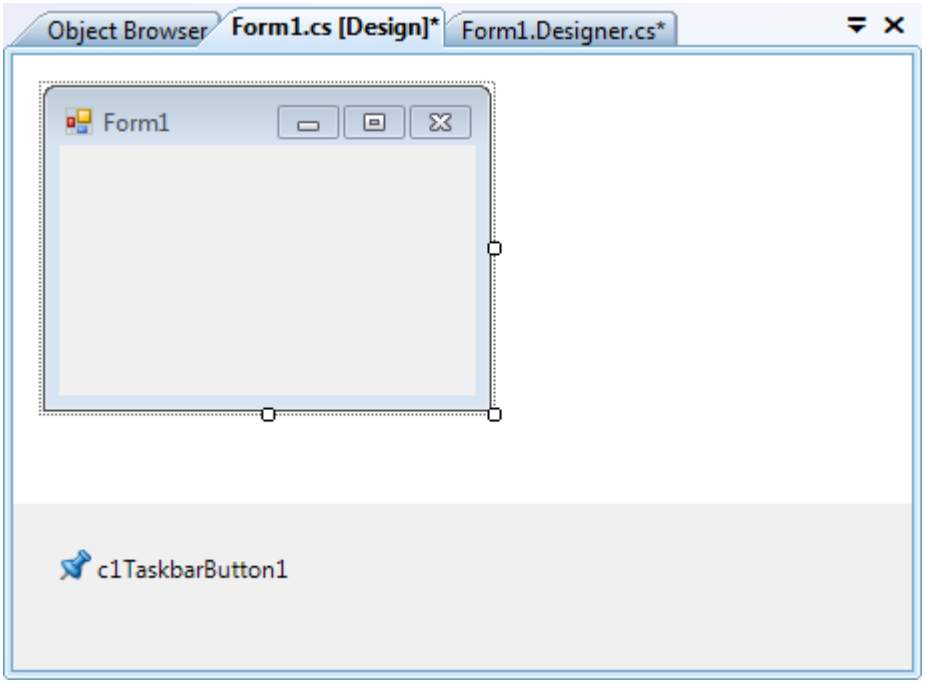
In this step you'll create a simple WinForms application and add the C1TaskbarButton component to the application.

To begin, complete the following steps:

1. From the **File** menu in Microsoft Visual Studio, select **New** and click **Project**. The **New Project** dialog box opens.
2. Under **Project Types**, choose either **Visual Basic Projects** or **Visual C# Projects**, and select **Windows Application** from the list of **Templates** in the right pane.
3. Enter or browse for a location for your application in the **Location** field and click **OK**.
   A new WinForms project is created in the specified location. In addition, a new form is displayed in the Designer view.
4. Navigate to the Toolbox and double-click the C1TaskbarButton item to add it to the application. The C1TaskbarButton component will appear in the component tray below the form.

What You've Accomplished

In this step, you created a new application and added the **C1TaskbarButton** control to the application. The project should appear as follows:

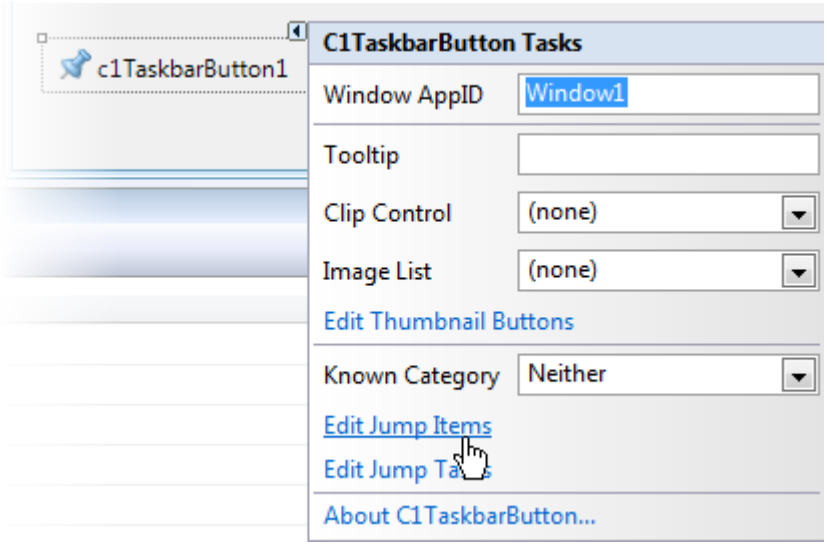


Step 2 of 4: Creating a Custom Jump List

In the last step, you added a C1TaskbarButton to the form, but it doesn't really do much at this point. You can do several things with it; you can add a progress indicator to it, create a jump list to it, or add a thumbnail toolbar to it. In this section of the quick start, we're going to add a jump list with jump links and jump tasks.

Complete the following steps:

1. Click **c1TaskbarButton1**'s smart tag to open the **C1TaskbarButton Tasks List**, and then click **Edit JumpItems**.



The **C1JumpList.Items Collection Editor** opens.

2. Click the **Add** drop-down arrow and select C1JumpLink from the list. In the properties grid, set the following properties:
   - Set the Title property to "Google Search".
   - Set the ApplicationPath property to the location of your Internet Explorer .exe file (by default, this is" C:\Program Files\Internet Explorer\iexplore.exe").
   - Set the Arguments property to "http://www.google.com".
   - Set the CustomCategory property to "Websites".
3. Now click the **Duplicate the selected item** button to duplicate the jump link, and then change the following properties in the properties grid:
   - Change the setting of the Title property to "Google News".
   - Change the setting of the ApplicationPath property to "http://news.google.com".
4. Click **OK** to close the collection editor.
5. Click **c1TaskbarButton1**'s smart tag to open the **C1TaskbarButton Tasks List**, and then click **Edit Jump Tasks**.

   The **C1JumpList.Task Collection Editor** opens.

6. Click Add to add a C1JumpTask to the jump list. Set the following properties.
   - Set the Title property to "Internet Explorer".
   - Set the ApplicationPath property to the location of your Internet Explorer .exe file (by default, this is" C:\Program Files\Internet Explorer\iexplore.exe").
7. Click **OK** to close the collection editor.

What You've Accomplished

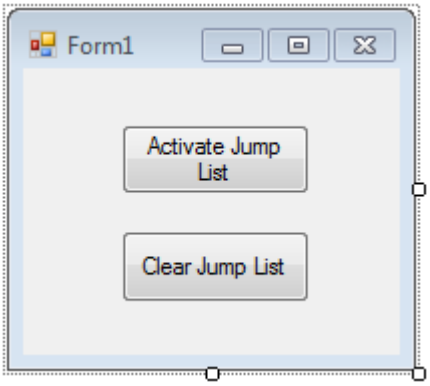
In this step, you added two jump links and two jump tasks to the jump list. In the next step, you'll learn how to apply changes to the jump list and how to clear the jump list.

Step 3 of 4: Using the Apply and Clear Methods

In the last step you created the jump list, but you still have to apply the jump list. In this step, you'll also learn how to clear jump lists.

Complete the following steps:

1. Navigate to the Toolbox and add two **Button** controls to your form. These will generically be named **button1** and **button2**.
2. Set **button1**'s **Text** property to "Activate Jump List".
3. Set **button2**'s **Text** property to "Clear Jump List".
4. Adjust the buttons on the form so that the text isn't clipped. The should look a little like this:



5. In Design view, double-click the **Activate Jump List** button to add the **button1_Click** event handler and then add the following code to it:

To write code in Visual Basic

```
Visual Basic
C1TaskbarButton1.JumpList.Apply()
```

To write code in C#

```
C#
c1TaskbarButton1.JumpList.Apply();
```

6. In Design view, double-click the **Clear Jump List** button to add a **button2_Click** event handler and then add the following code to it:

To write code in Visual Basic

```
Visual Basic
C1TaskbarButton1.JumpList.ClearTasksAndCustomCategories()
```

To write code in C#

```
C#
c1TaskbarButton1.JumpList.ClearTasksAndCustomCategories();
```

What You've Accomplished

In this step, you added code to the project that will allow you to apply or clear the jump list at the click of a button. In the next step, you'll run the project and see the result of your work.

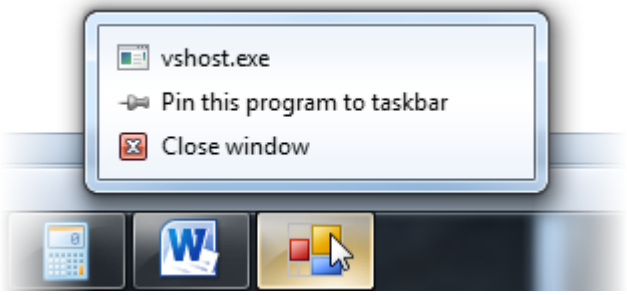
Step 4 of 4: Running the Project

In the previous steps, you added a C1TaskbarButton control to your project, added a jump list to the control, and used

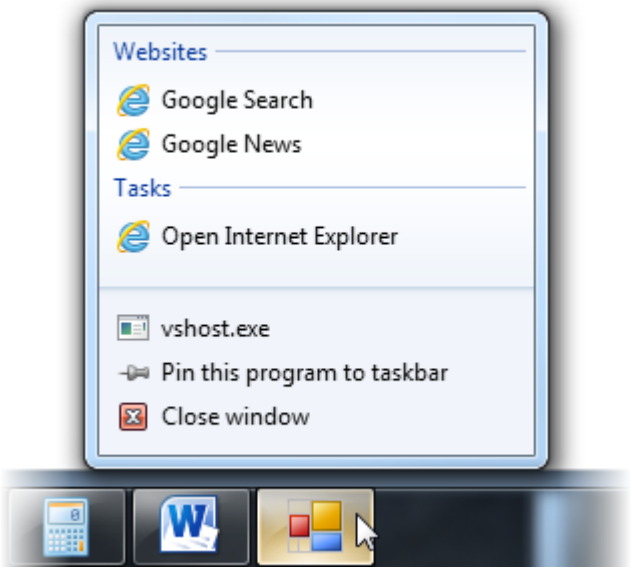
methods that will apply changes and clear changes to the control. In this step, you'll run the project and see the results of this quick start.

Complete the following steps:

1. Press **F5** to build the project.
2. Right-click the taskbar button and observe that only the default list items appear.



3. On the form, click the **Activate Jump List** button.
4. Right-click the taskbar button and observe that the items you added in this tutorial appear.



5. On the form, click the **Clear Jump List** button.
6. Right-click the taskbar button and observe that the items you added in this tutorial topic have been cleared and, once again, only the default items appear.

What You've Accomplished

You've completed the C1TaskbarButton control's quick start tutorial. If you'd like to learn more about this control, see the C1TaskbarButton Task-Based Help and the Working with C1TaskbarButton topics.

Windows 7 Control Pack for WinForms Components

Windows 7 Control Pack for WinForms consists of the following controls and components, which provide functionality for creating a Windows 7-style application.

The C1TaskbarButton Component

C1TaskbarButton works with Windows 7 taskbar extensions. You can adjust the application's Jump List, thumbnail toolbar, icon overlay, and a progress bar displayed on the taskbar button. Also, you can select a particular control on a form to use its contents as the thumbnail image.

The C1TaskDialog Control

C1TaskDialog is a powerful replacement for the standard message box under Windows Vista or newer operating systems. Similar to other standard dialog boxes (such as **OpenFileDialog**) you can set simply a few properties, then call to the Show method to display the dialog box at run-time.

C1TaskbarButton Component

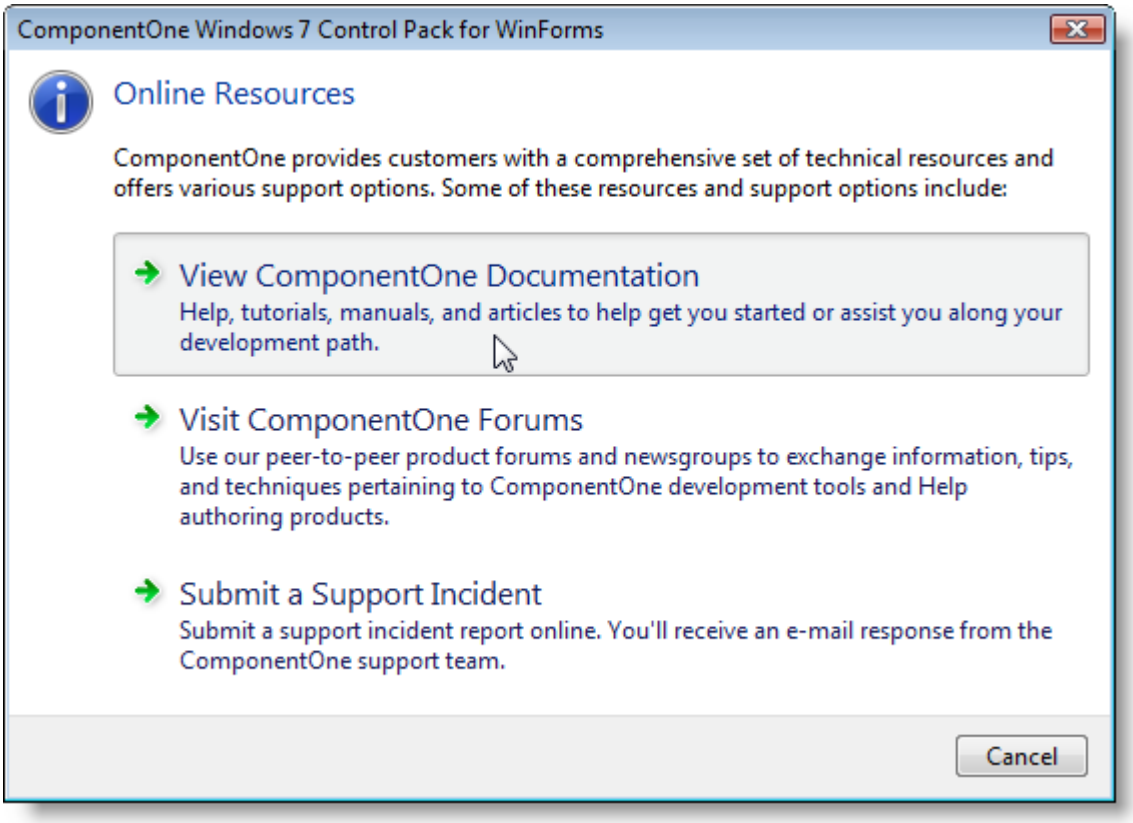
The C1TaskbarButton component works with Windows 7 taskbar extensions. You can adjust the application's Jump List, thumbnail toolbar, icon overlay, and a progress bar displayed on the taskbar button. Also, you can select a particular control on a form to use its contents as the thumbnail image.

C1TaskDialog Component

You can use the C1TaskDialog control as you would any other dialog box. A dialog box is a secondary window that allows users to perform a command, asks users a question, or provides users with information or progress feedback.

Dialog boxes consist of a title bar (to identify the command, feature, or program where a dialog box came from), an optional main instruction (to explain the user's objective with the dialog box), various controls in the content area (to present options), and commit buttons (to indicate how the user wants to commit to the task).

For example, a C1TaskDialog control featuring Command Links:



C1TaskDialog creates task dialog boxes. Task dialog boxes, like those you can create with C1TaskDialog, typically consist of the following parts:

- A title bar to identify the application or system feature where the dialog box came from.
- A main instruction, with an optional icon, to identify the user's objective with the dialog box.
- A content area for descriptive information and controls.
- A command area for commit buttons, including a **Cancel** button, and optional **More options** and **Don't show this item again** controls.
- A footnote area for optional additional explanations and help, typically targeted at less experienced users.

Design-Time Support

Windows 7 Control Pack for WinForms provides visual editing to make it easier to create a Windows 7 application. The following sections describe how to use **Windows 7 Control Pack for WinForms'** design-time environment to configure the **Windows 7 Control Pack for WinForms** controls:

Smart Tags and Tasks Menus

You can invoke each control's tasks menu by clicking on the smart tag (▶) in the upper-right corner of the control. A smart tag represents a short-cut tasks menu that provides the most commonly used properties in each control. For more information on how to use the tasks menu for each control in **Windows 7 Control Pack for WinForms**, see C1TaskDialog Tasks Menu and C1TaskbarButton Tasks Menu.

Properties Window

You can also easily configure **Windows 7 Control Pack for WinForms** at design time using the Properties window in Visual Studio. You can access the Properties window by right-clicking the control and selecting **Properties**.

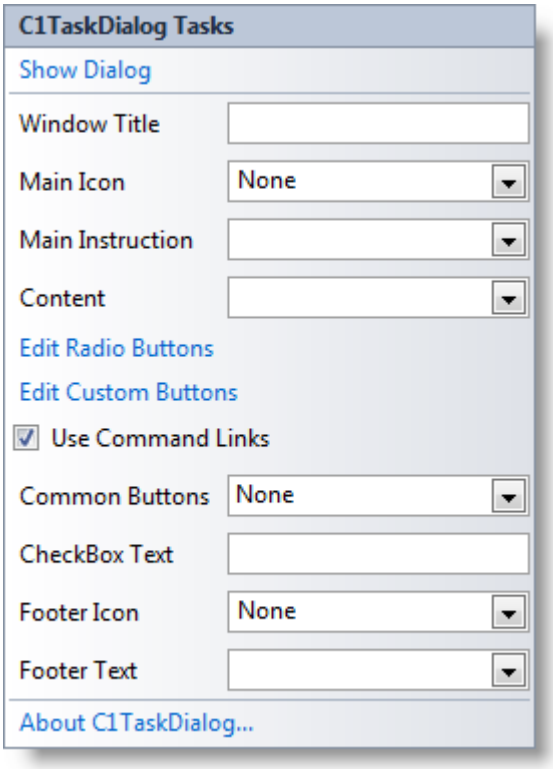
Windows 7 Control Pack for WinForms Smart Tags

The C1TaskDialog and C1TaskbarButton controls include a smart tag. A smart tag represents a short-cut tasks menu that provides the most commonly used properties in each control. You can invoke the tasks menu by clicking on the smart tag (▶) in the upper-right corner of each control. For more information on how to use the tasks menu for each control in **C1Schedule**, see C1TaskDialog Tasks Menu and C1TaskbarButton Tasks Menu.

C1TaskDialog Tasks Menu

In the **C1TaskDialog Tasks** menu you can quickly and easily customize the appearance and behavior of the C1TaskDialog component's dialog box.

To access the **C1TaskDialog Tasks** menu, click on the smart tag (▶) in the upper right corner of the control. This will open the **C1TaskDialog Tasks** menu.



Options include:

- **Show Dialog**
  Clicking the **Show Dialog** option displays the current dialog box. This option is useful as it allows you to preview the appearance of the dialog box without running your application.
- **Window Title**
  Enter text in this text box to set the text that appears on the caption bar of the dialog box.
- **Main Icon**
  This option sets the main icon displayed in the dialog box and in the upper-left corner of the dialog box's caption bar. Click this item's drop-down allow and choose an option. Options include **None** (default), **Shield**, **Information**, **Error**, and **Warning**.
- **Main Instruction**
  Enter text in this text box to set the larger heading text that will be displayed at the top of the dialog box's content area.
- **Content**
  Text entered in this area will be displayed below the Main Instruction and will typically consist of more detailed information or instructions. You can enter HTML links in this section as well.
- **Edit Radio Buttons**
  Clicking this item will open the **C1TaskDialog.RadioButtons Collection Editor**. In this dialog box you can add radio buttons to the dialog box and customize the text displayed next to each button. If you add a radio button the dialog box will be previewed, to close the preview click in Visual Studio. See RadioButtons Collection Editor for more information.
- **Edit Custom Buttons**
  Clicking this item will open the **C1TaskDialog.CustomButtons Collection Editor**. In this dialog box you can add custom buttons to the dialog box and customize the text displayed next to each button. See CustomButtons Collection Editor for more information.
- **Use Command Links**
  When this check box is checked (default) custom buttons will appear as command links rather than standard buttons. Command links have a clean, lightweight appearance that allows for descriptive labels, and are displayed with either a standard arrow or custom icon, and an optional supplemental explanation. See Command Links for more information.

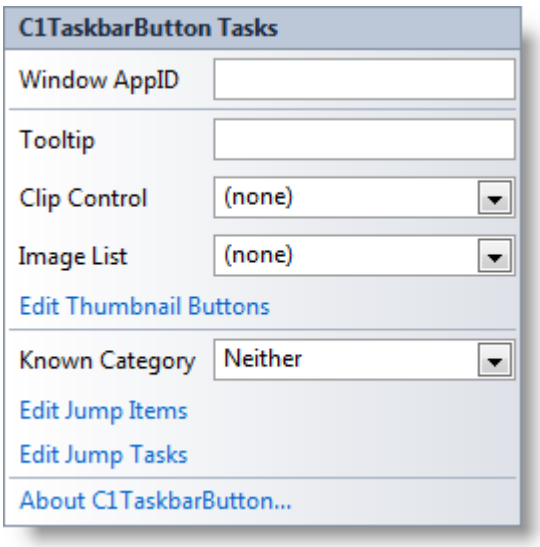
- **Common Buttons**
  Select this option to add standard buttons to the dialog box. Click the drop-down arrow and check the check boxes for the buttons you wish to include. Available buttons include **Ok**, **Yes**, **No**, **Cancel**, **Retry**, and **Close**. The default option is **None**.
- **CheckBox Text**
  Text entered in this text box will be displayed next to a check box in the dialog box. If no text is entered (default), the check box will not be displayed in the dialog box.
- **Footer Icon**
  This option sets the icon displayed in the footer area of the dialog box. Click this item's drop-down arrow and choose an option. Options include **None** (default), **Shield**, **Information**, **Error**, and **Warning**.
- **Footer Text**
  Text entered in this section will be displayed in the footer area of the dialog box.
- **About C1TaskDialog**
  Clicking the **About** item displays the C1TaskDialog control's **About** dialog box, which is helpful in finding the build number of the control.

You can also access some of these options in the component's context menu and more options in the Properties window.

C1TaskbarButton Tasks Menu

In the **C1TaskbarButton Tasks** menu, you can quickly and easily edit the appearance and behavior of the C1TaskbarButton component..

To access the **C1TaskbarButton Tasks** menu, click on the smart tag (▶) in the upper right corner of the control. This will open the **C1TaskbarButton Tasks** menu.



Options include:

- **Window AppID**

  Specifies the Application User Model ID for an individual window. Application User Model IDs are used extensively by the taskbar in Windows 7 and later systems to associate processes, files, and windows with a particular application.

- **Tooltip**

Specifies the ToolTip that will be displayed for an item in the taskbar. ToolTips are typically displayed when a user hovers the mouse over an item.

- **Clip Control**

  Specifies a control to clip and display as a thumbnail image. If you choose a control from the drop-down box, a portion of it will be restricted to use as an image.

- **Image List**

  Here you can select an image collection that might typically be used by another control such as a **ListView**, **TreeView**, or **ToolStrip** control.

- **Edit Thumbnail Buttons**

  Clicking this item will open the **C1Thumbnail. Buttons Collection Editor**. In this dialog box you can add and customize the appearance and behavior of thumbnail buttons to be used in the taskbar button flyout. See Buttons Collection Editor for more information.

- **Known Category**

  In this drop-down box you can select the type of known category to display. Options include **Neither** (default), **Recent**, and **Frequent**.

- **Edit Jump Items**

  Clicking this item will open the **C1JumpList. Items Collection Editor**. In this dialog box you can add and customize the appearance and behavior of C1JumpTask and C1JumpLink items that appear in the jump list. See Items Collection Editor for more information.

- **Edit Jump Tasks**

  Clicking this item will open the **C1JumpList. Tasks Collection Editor**. In this dialog box you can add and customize the appearance and behavior of task (shortcut) items that appear in the jump list. See Tasks Collection Editor for more information.

- **About C1TaskbarButton**

  Clicking the **About** item displays the C1TaskbarButton control's **About** dialog box, which is helpful in finding the build number of the control.

You can also access some of these options in the component's context menu and more options in the Properties window.

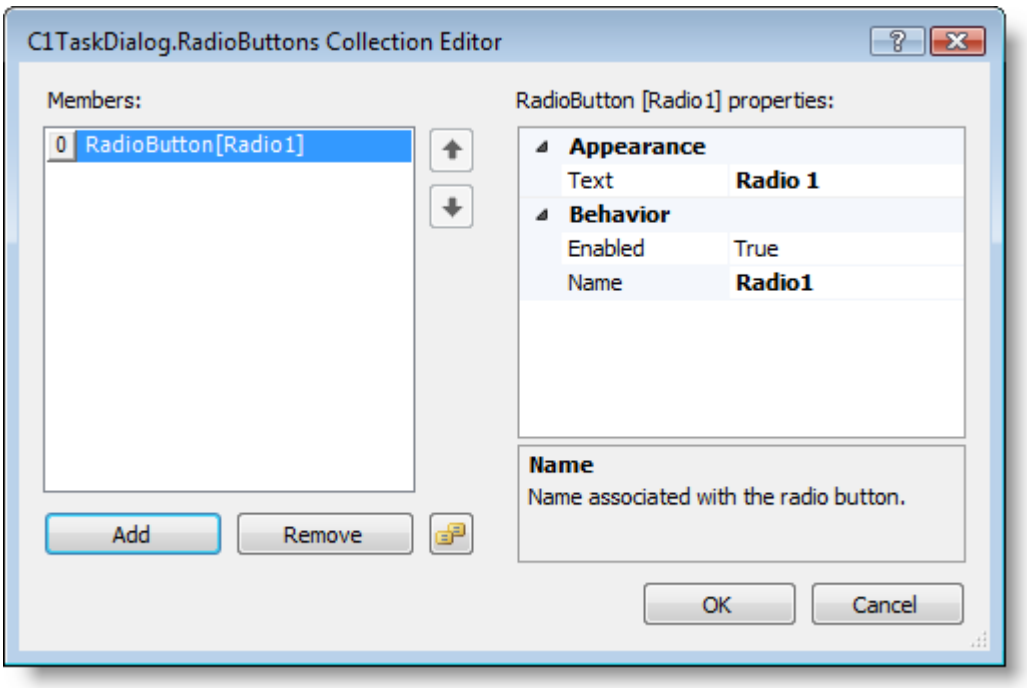
C1TaskDialog Collection Editors

The following topics detail collection editors that you can use to customize the C1TaskDialog component. Each of these editors can be accessed from the C1TaskDialog control's Tasks menu, context menu, and Properties window.

RadioButtons Collection Editor

In the **C1TaskDialog.RadioButtons Collection Editor** dialog box you can add radio buttons to the C1TaskDialog dialog box and customize the text displayed next to each button. To access the **RadioButtons Collection Editor**, select the **Edit Radio Buttons** link from the C1TaskDialog control's Tasks menu, context menu, or below the Properties window, or click the ellipses button next to the RadioButtons property in the Properties window.

The **RadioButtons Collection Editor** appears similar to the following:



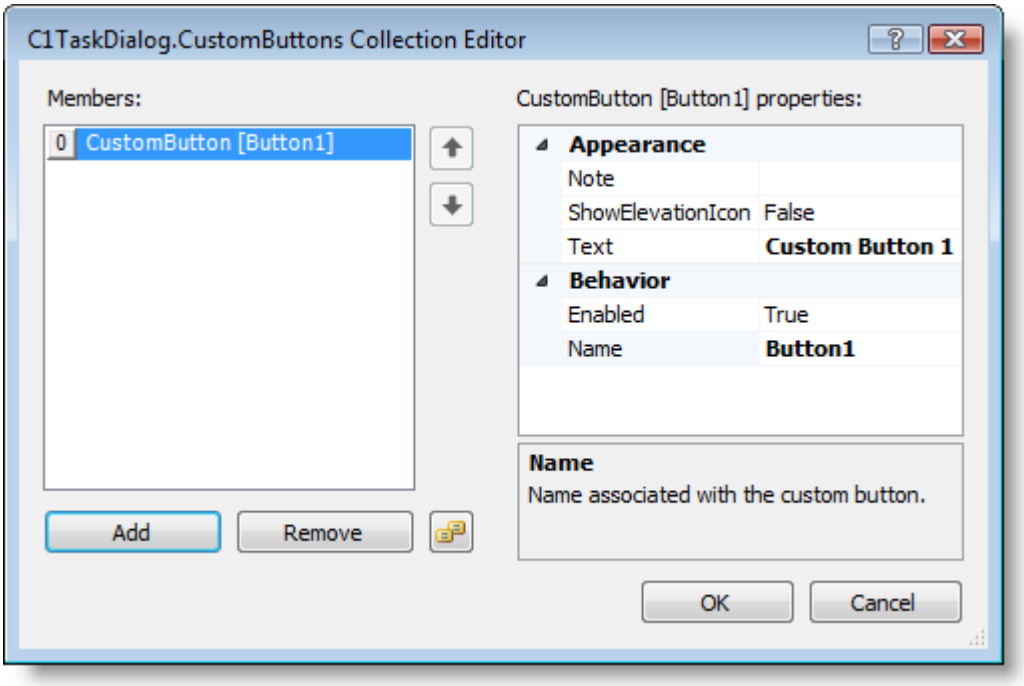
You can select an item in the Members list or, using the buttons below the list, you can add and remove radio buttons or copy existing radio buttons. For each radio button you can set or change the Text that appears next to the button, enable or disable the button, and set or change the name of the button.

If you add a radio button the dialog box with your changes will be automatically previewed. To close the preview click elsewhere in Visual Studio.

CustomButtons Collection Editor

In the **C1TaskDialog.CustomButtons Collection Editor** dialog box you can add custom buttons to the dialog box and customize the text displayed next to each button. To access the **CustomButtons Collection Editor**, select the **Edit Custom Buttons** link from the C1TaskDialog control's Tasks menu, context menu, or below the Properties window, or click the ellipses button next to the CustomButtons property in the Properties window.

The **CustomButtons Collection Editor** appears similar to the following:



You can select an item in the Members list or, using the buttons below the list, you can add and remove buttons or copy existing buttons. For each button you can set or change the Text that appears next to the button, determine whether to show the elevation icon, add a note to add an extended text description of the button, enable or disable the button, and set or change the name of the button.

The text you set in the **Note** property will only be displayed if the button is set to appear like a command link and not a standard button. See Command Links for more information.

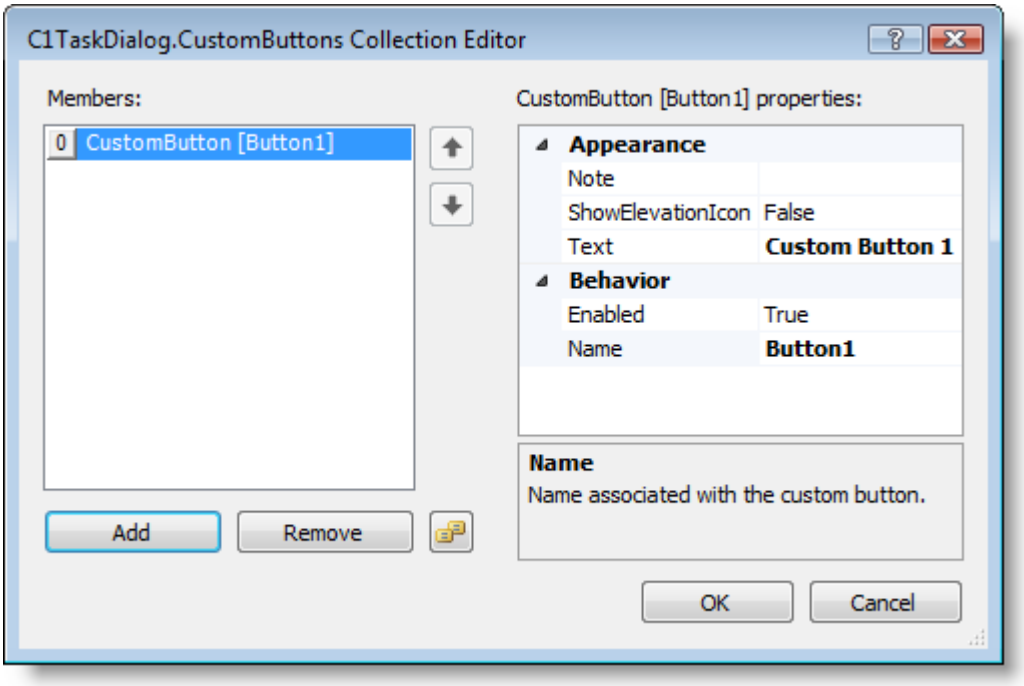
C1TaskbarButton Collection Editors

The following topics detail collection editors that you can use to customize the C1TaskbarButton component. Each of these editors can be accessed from the C1TaskbarButton control's Tasks menu, context menu, and Properties window.

Buttons Collection Editor

In the **C1TaskbarButton.Buttons Collection Editor** dialog box you can add thumbnail buttons to the taskbar and customize the appearance and behavior of each button. To access the **Buttons Collection Editor**, select the **Edit Thumbnail Buttons** link from the C1TaskbarButton control's Tasks menu, context menu, or below the Properties window, or click the ellipses button next to the Buttons property in the Properties window.

The **Buttons Collection Editor** appears similar to the following:



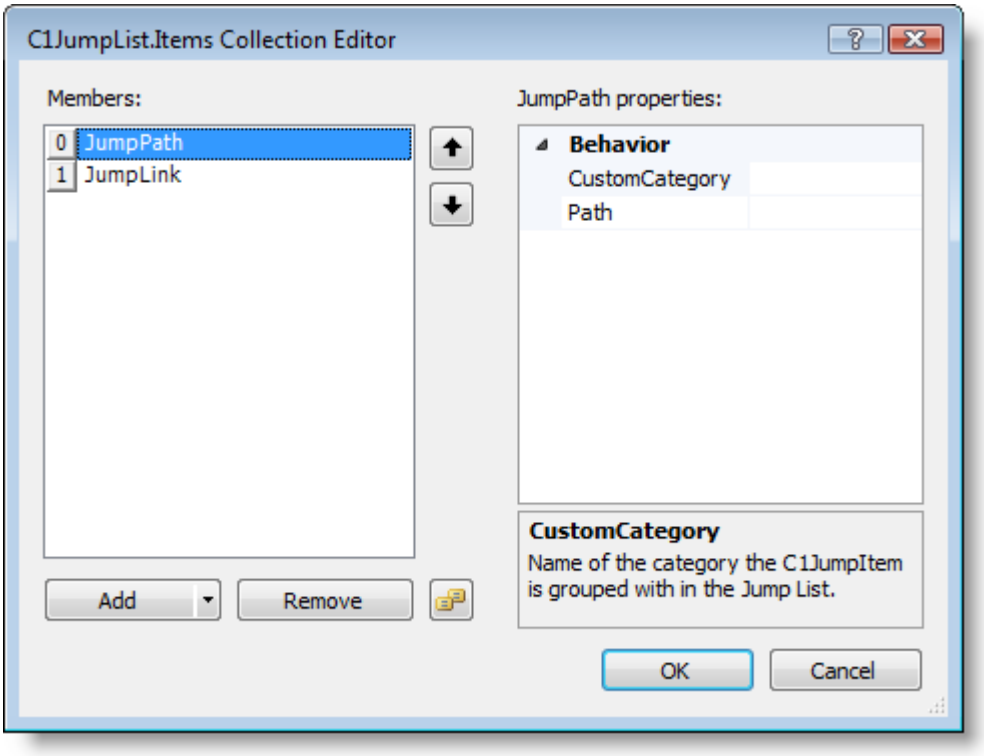
You can select an item in the Members list or, using the buttons below the list, you can add and remove buttons or copy existing buttons. For each button you can set or change icon, image, and background for the button, the ToolTip that appears on mouse hover, the name of the button, and if it's enabled, interactive, and visible.

Items Collection Editor

In the **C1JumpList. Items Collection Editor** dialog box you can add and customize the appearance and behavior of C1JumpPath and C1JumpLink items that appear in the jump list. To access the **Items Collection Editor**, select the **Edit Jump Items** link from the C1TaskbarButton control's Tasks menu, context menu, or below the Properties window, or click the ellipses button next to the Items property in the Properties window.

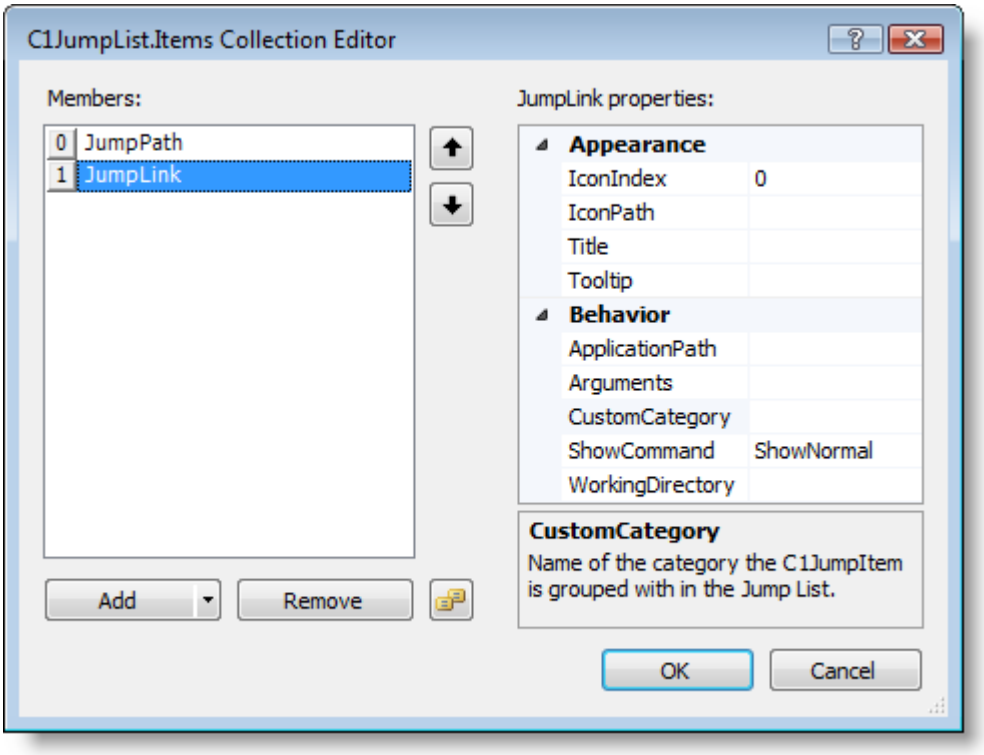
You can select an item in the Members list or, using the buttons below the list, you can add and remove buttons or copy existing buttons. The **Add** button is a split button, to access all options on the **Add** button click the drop-down arrow on the button.

The **Items Collection Editor** appears similar to the following when a C1JumpPath is selected:



You can customize the C1JumpPath by setting the custom category the item is grouped in and the path to the file to be included.

The **Items Collection Editor** appears similar to the following when a C1JumpLink is selected:

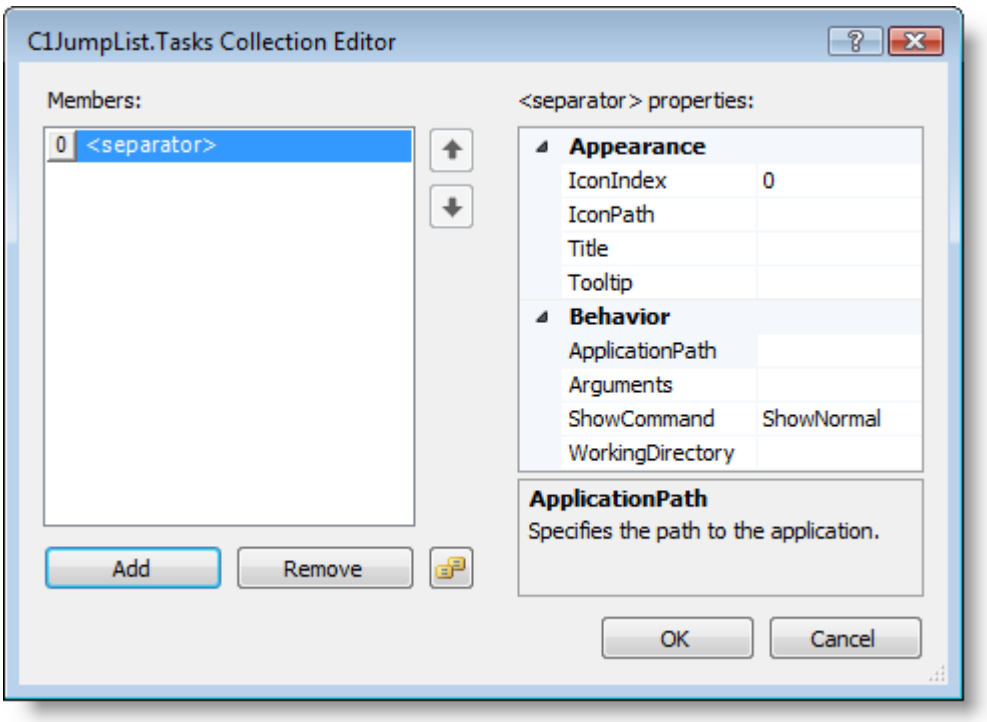


For each C1JumpLink you can set or change icon and title for the list, the ToolTip that appears on mouse hover, the application path, arguments, the category it is included in, how it should be shown, and the working directory of the application on startup.

Tasks Collection Editor

In the **C1JumpList.Tasks Collection Editor** dialog box you can add and customize the appearance and behavior of task (shortcut) items that appear in the jump list. To access the **Tasks Collection Editor**, select the **Edit Jump Tasks** link from the C1TaskbarButton control's Tasks menu, context menu, or below the Properties window, or click the ellipses button next to the Tasks property in the Properties window.

The **Tasks Collection Editor** appears similar to the following:



You can select an item in the Members list or, using the buttons below the list, you can add and remove items or copy existing items. For each task you can set or change icon, image, and title for the item, the ToolTip that appears on mouse hover, the application path, arguments, the category it is included in, how it should be shown, and the working directory of the application on startup.

Working with Windows 7 Control Pack for WinForms

The **Windows 7 Control Pack for WinForms** allows you to extend the behavior of the standard task bar and dialog box in Microsoft Windows 7. The following topics explain the main aspects of the **Windows 7 Control Pack for WinForms** controls.

Working with C1TaskDialog

C1TaskDialog is a powerful replacement for the standard message box under Windows Vista or newer operating systems. The following topics explain the main aspects of the C1TaskDialog control.

C1TaskDialog Operating System Compatibility

C1TaskDialog is only supported on Windows Vista or later. The control will simply not display on older versions of Windows and will not throw an exception. If the C1TaskDialog control is included in an application that may be run on previous versions of Windows – Windows XP, for example – you may want to specify an alternative dialog box so that important messages are displayed. You can do so by using the IsPlatformSupported property to detect the user's operating system.

For example:

To write code in Visual Basic

Visual Basic

```
If C1TaskDialog.IsPlatformSupported Then
    c1TaskDialog1.Show()
Else
    ' Show a brief message with simple choice of the action
    Dim res As DialogResult = MessageBox.Show("Text", "Caption",
MessageBoxButtons.YesNoCancel)
    If res = DialogResult.Yes Then
        ....
    End If
End If
```

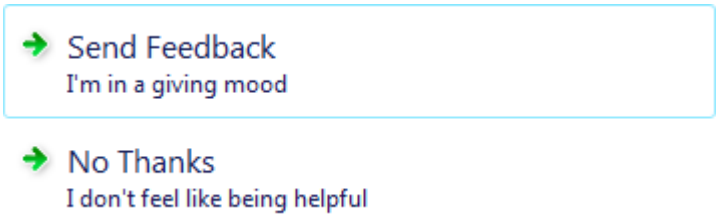
To write code in C#

C#

```
if (C1TaskDialog.IsPlatformSupported)
    c1TaskDialog1.Show();
else
{
    // Show a brief message with simple choice of the action
    DialogResult res = MessageBox.Show("Text", "Caption",
MessageBoxButtons.YesNoCancel);
    if (res == DialogResult.Yes)
    {
        ...
    }
}
```

Command Links

Command links are customized button that provide explanations about various choices. Command links have a clean, lightweight appearance that allows for descriptive labels, and are displayed with either a standard arrow or custom icon, and an optional supplemental explanation.



For custom buttons to be displayed as command links set the UseCommandLinks property to **True** (default). If you do not want the command link to include an icon, set the NoIconOnCommandLinks property to **False**. To make a command link the default button, you can use the SetDefaultButton method to indicate the name of the button. See the Setting the Default Button topic for an example.

Command links are similar to radio buttons in that they are used to select from a set of mutually exclusive, related choices. Like radio buttons, command links are always presented in sets, never individually. In appearance, command links have the lightweight appearance similar to regular links, without a frame or other strong click affordance.

Command links are also similar to command buttons, in that they can be the default "command button" and they can have an access key assigned. Like commit buttons, on click they either close the window (for dialog boxes) or advance to the next page (for wizards and pages flows).

Access Keys

An access key is an underlined character in the text of a menu, menu item, or the label of a control such as a button. With an access key, the user can "click" a button by pressing the ALT key in combination with the predefined access key. For example, if a button runs a procedure to print a form, and therefore its Text property is set to "Print," adding an ampersand before the letter "P" causes the letter "P" to be underlined in the button text at run time. The user can run the command associated with the button by pressing ALT+P. You cannot have an access key for a control that cannot receive focus.

You can easily add access keys to buttons and command link buttons in a **C1TaskDialog** dialog box. To do so, you would need to set the button's **Text** property to a string that includes an ampersand (&) before the letter that will be the shortcut.

Note: To include an ampersand in a caption without creating an access key, include two ampersands (&&). A single ampersand is displayed in the caption and no characters are underlined.

See the Setting the Access Key topic for an example.

Dialog Box Icons

The **C1TaskDialog** control includes several built-in icons that can be displayed in the header area or the footer area of the dialog box. The following common icons are included:

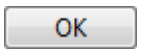
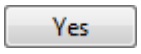
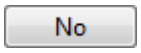
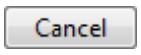
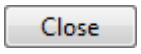
Icon	Name	Description
None	None	Displays no icons.

	Shield	A shield icon appears in the task dialog box.
	Information	An icon consisting of a lowercase letter "i" in a circle appears in the task dialog box.
	Error	A stop-sign icon appears in the task dialog box.
	Warning	An exclamation-point icon appears in the task dialog box.

To have an icon appear in the header area and in the caption bar, set the MainCommonIcon property to one of the above options. To set the footer icon, set the FooterCommonIcon property to one of the above icons. Note that if you want to add a custom icon to the header or footer areas, you can instead set the MainCustomIcon and FooterCustomIcon properties to icon images of your choice.

Common Buttons

The **C1TaskDialog** control includes several built-in buttons that can be included in the bottom right corner of the dialog box. To customize the dialog box with buttons, you can set the CommonButtons property to one or more of the following values:

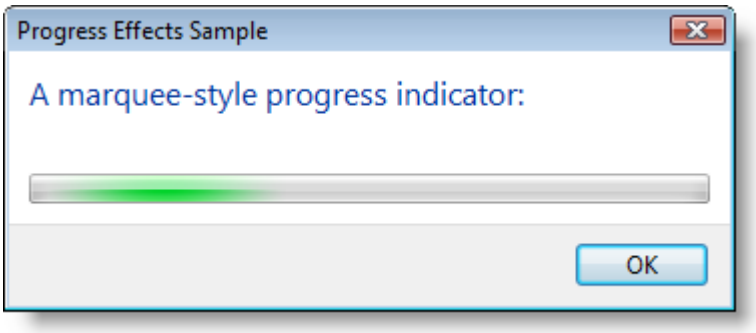
Button	Name	Description
None	None	Displays no buttons.
	Ok	Displays the **Ok** button.
	Yes	Displays the **Yes** button.
	No	Displays the **No** button.
	Retry	Displays the **Retry** button.
	Cancel	Displays the **Cancel** button.
	Close	Displays the **Close** button.

To set the default button, you can use the SetDefaultButton method to indicate the name of the button. See the Setting the Default Button topic for details.

Progress Bar

The **C1TaskDialog** control includes a progress bar, C1ProgressBar, that can be included in a **C1TaskDialog** dialog box. A progress bar can be used to indicate the progress of a lengthy operation. The progress bar consists of an area that is filled from left to right as an operation progresses.

For example:



A **C1ProgressBar** control visually indicates the progress of a lengthy operation in one of two styles:

- A continuous bar that fills in from left to right.
- A block that scrolls across a progress bar in a marquee fashion.

By default the section of the progress bar that is being filled appears green, but it can appear yellow when the progress bar is paused, or red if there was an error and the progress bar is stopped. The State property sets the type and state of the progress bar.

You can set the State property to one of the following options:

State	Description
Indeterminate	The progress is indeterminate (marquee).
Normal	Normal progress is displayed.
Error	An error occurred (red).
Paused	The operation is paused (yellow).

A progress bar is typically used when an application performs tasks such as copying files or printing documents. Users of an application might consider an application unresponsive if there is no visual cue. By using the progress bar in your application, you alert the user that the application is performing a lengthy task and that the application is still responding.

Working with C1TaskbarButton

C1TaskBarButton manages the application button on the Windows 7 taskbar. With it, you can create jump lists, add thumbnail buttons, and display progress indicators on the taskbar with this straightforward, developer-friendly component.

There are essentially three elements to the C1TaskbarButton control. The first is the taskbar button itself, which can be modified with progress indicators and icon overlays; the second is the jump list, which appears when users right-click the taskbar button; and the third is the thumbnail, which appears when users hover over the taskbar button.

Before you begin working with the different elements of the C1TaskbarButton control, it is recommended that you read the C1TaskbarButton Operating System Compatibility topic.

C1TaskbarButton Operating System Compatibility

C1TaskbarButton is only supported by Windows 7. The control will not display on older versions of Windows, but it will not throw an exception if the user is using an earlier version of Windows. If the C1TaskDialog control is included in an application that may be run on previous versions of Windows – Windows XP or Vista, for example – you may want to specify an alternative dialog box so that important messages are displayed. You can do so by using the

IsPlatformSupported property to detect the user's operating system.

For example:

To write code in Visual Basic

```
Visual Basic
If C1TaskbarButton.IsPlatformSupported Then

Else
    ' Show a brief message with simple choice of the action
    Dim res As DialogResult = MessageBox.Show("Text", "Caption",
MessageBoxButtons.YesNoCancel)
    If res = DialogResult.Yes Then
        ....
    End If
End If
```

To write code in C#

```
C#
if (C1TaskbarButton.IsPlatformSupported)

else
{
    // Show a brief message with simple choice of the action
    DialogResult res = MessageBox.Show("Text", "Caption",
MessageBoxButtons.YesNoCancel);
    if (res == DialogResult.Yes)
    {
        ...
    }
}
```

Taskbar Button Elements

The taskbar button is simply a button that can be clicked to open an application, display a thumbnail, or open a jump list. This section of the Help only covers the face of the taskbar button. To learn more about the jump lists and thumbnails that can be constructed for the C1TaskbarButton control, see Jump List Elements and Thumbnail Elements.

Taskbar Button Progress Indicator

The C1TaskbarButton control includes a progress indicator. The progress indicator is visible on the taskbar button; it can be used to communicate the progress of an operation, or it can be used to convey that an ongoing operation is in progress. The progress bar always fills from the right-to-left on the taskbar button.

States

The progress bar can be set to one of four states, which are as follows:

State	Description	Example
Indeterminate	The Indeterminate state is what's commonly referred to as the "marquee" style.. It indicates an ongoing process by continuously animating the progress bar indicator rolling across the screen. The progress indicator in the Indeterminate state will always be green.	
Normal	The Normal state uses a green progress indicator that only advances when a value has changed. It indicates to users that the operation is still processing as expected whilst giving them an indication of the progress of the operation.	
Error	The Error state changes the color of the progress bar indicator to red, which indicates to the user that there is an issue with the operation.	
Paused	The Paused state changes the color of the progress bar's indicator to yellow, which indicates that the operation is currently paused. This is commonly used when a user initiates a pause or when the a user must interact with something to advance the operation.	

Maximum Value

You can set a value of the complete operation by setting the Maximum property to a number. By default, the Maximum property is set to 100.

Value

The progress bar's value is set by the Value property. If you wanted to set the value to the value of a trackbar, for example, you would add the following code in the trackbar's **Scroll** event:

To write code in Visual Basic

```
Visual Basic
C1TaskbarButton1.ProgressIndicator.Value = trackBar1.Value
```

To write code in C#

```
C#
c1TaskbarButton1.ProgressIndicator.Value = trackBar1.Value;
```

Show and Hide Methods

The C1ProgressIndicator class contains Show and Hide methods, making it simple for you to make progress bars visible and invisible to users.

To write code in Visual Basic

Visual Basic

```
Private Sub progressCheckBox_CheckedChanged(sender As Object, e As EventArgs)
       If progressCheckBox.Checked Then
              taskbarButton.ProgressIndicator.Show()
       Else
              taskbarButton.ProgressIndicator.Hide()
       End If
End Sub
```

To write code in C#

C#

```
private void progressCheckBox_CheckedChanged(object sender, EventArgs e)
{
     if (progressCheckBox.Checked)
     {
          taskbarButton.ProgressIndicator.Show();
     }
     else
     {
          taskbarButton.ProgressIndicator.Hide();
     }
}
```

Taskbar Button Overlay

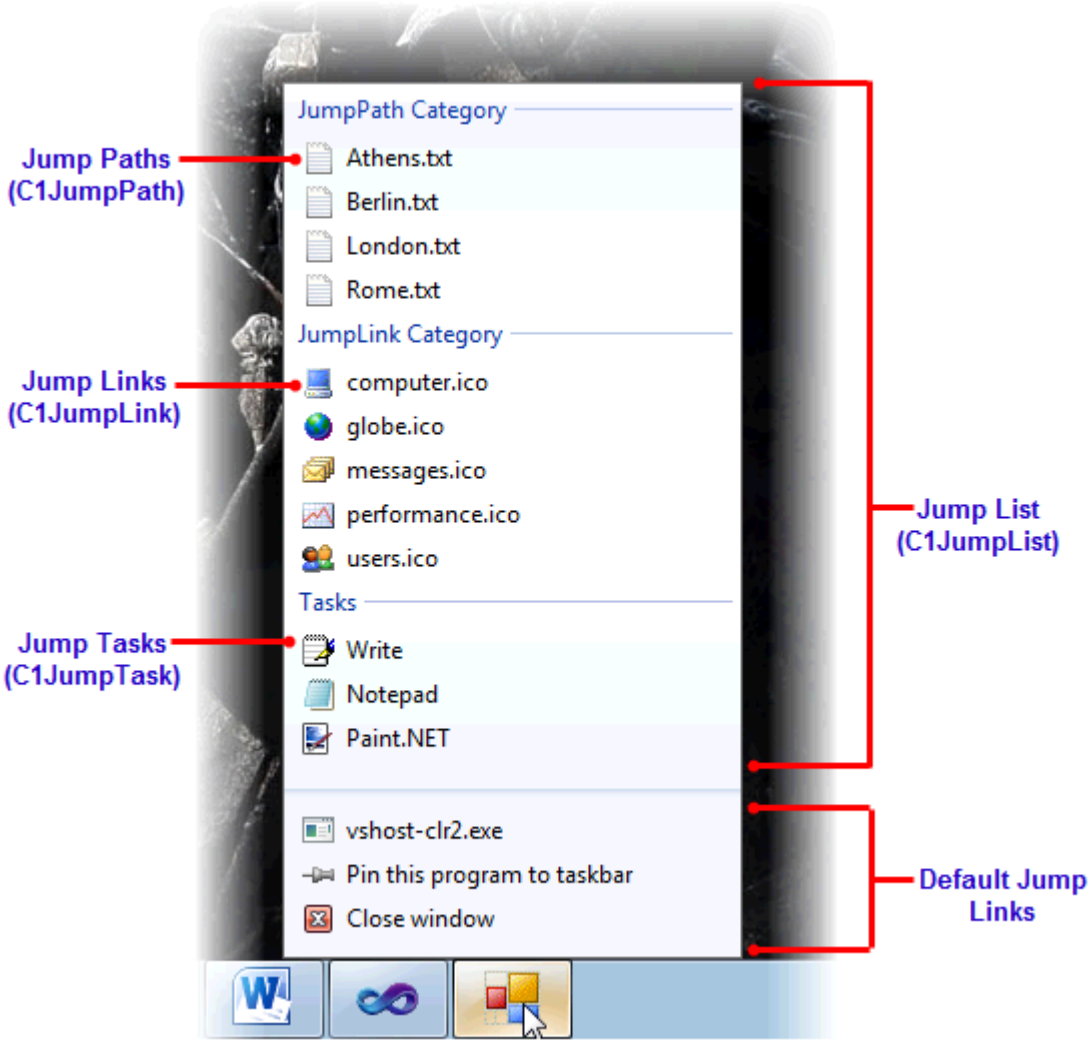
The overlay image is used to communicate information with a user about the state of an application. The C1TaskbarButtons overly icon can be set at design time in the **Properties** window or can be set via code for run time interaction. It is more common to see this property set when a specific event occurs at run time.

The OverlayIcon property takes type System.Drawing.Icon and requires the size of the icon to be no more than 16x16 pixels.

For task-based help about overlays, see Adding an Overlay Image to the Taskbar Button.

Jump List Elements

A jump list is a collection of jumps (or links) to documents, commands, files, programs, or links that can accessed by right-clicking a taskbar button. In the C1TaskbarButton component, the C1JumpList class represents a jump list. A jump list may contain the following elements:



C1JumpList

There are several things you can do to modify the jump list. For example, you can set the maximum number of items that will appear on the jump list using the MaxSlots property. You may also set the known category section and the position of the known category section via, respectively, the KnownCategory property and the KnownCategoryPosition property.

There are three settings for the KnownCategory property: **Recent**, **Frequent**, and **Neither**. If you choose **Recent**, the list will display a series of links to files that you've recently used. If you choose **Frequent**, the list will display a series of links to applications and files you frequently use. If you select **Neither**, the user will never see a Recent or a Frequent category associated with your. list.

C1JumpPaths, C1JumpLinks, and C1JumpTasks

C1JumpPaths and C1JumpLinks are both of the C1JumpItem class, whereas a C1JumpTask is represented by the C1JumpTask class. The first two, C1JumpPaths and C1JumpLinks, are added to the C1JumpList by way of the C1JumpLists Tasks property, while C1JumpTasks must be added using the C1JumpLists Tasks property.

To read more about C1JumpPaths, C1JumpLinks, and C1JumpTasks, see C1JumpPath Basics, C1JumpLink Basics, and C1JumpTask Basics.

Applying Changes to the Jump List

To apply changes to the jump list, you will need to call the C1JumpList.Apply method.

To write code in Visual Basic

```
Visual Basic
C1TaskbarButton1.JumpList.Apply()
```

To write code in C#

```
C#
c1TaskbarButton1.JumpList.Apply();
```

C1JumpPath Basics

A jump path, represented by the C1JumpLink class and inheriting from the C1JumpItem class, is reference to a file that can be accessed and opened from the taskbar's jump list.

C1JumpPaths can be linked to any type of file; the only requirement is that your project be registered to handle the file type. To register the file type, use the C1TaskbarButton's RegisterFileAssociations method, which takes two overloads. The following method registers a .png file and commands the link to open Paint.NET to display the file:

To write code in Visual Basic

```
Visual Basic
C1TaskbarButton1.JumpList.Apply()
```

To write code in C#

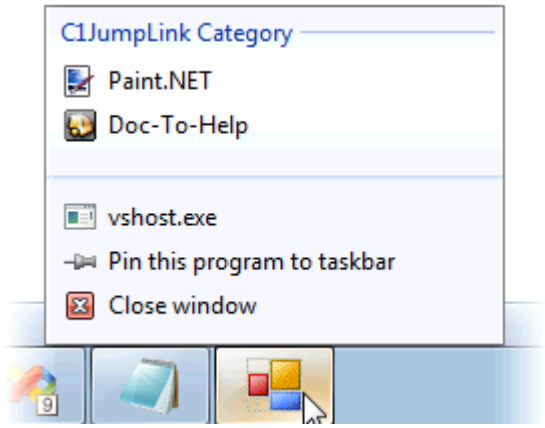
```
C#
c1TaskbarButton1.JumpList.Apply();
```

C1JumpPath has two settable properties: Path and CustomCategory. Path simply sets the location of the file, and CustomCategory specifies the category the jump path will appear in on the jump list. If the CustomCategory property is not set, the paths will just appear in the Recent or Frequently Used categories.

By default, the C1JumpPaths icon will be the Windows default icon for that type of file, but you can change that by specifying an icon in the RegisterFileAssociations method's *defaultIcon* string. The title of the C1JumpPath will be the same as the filename.

C1JumpLink Basics

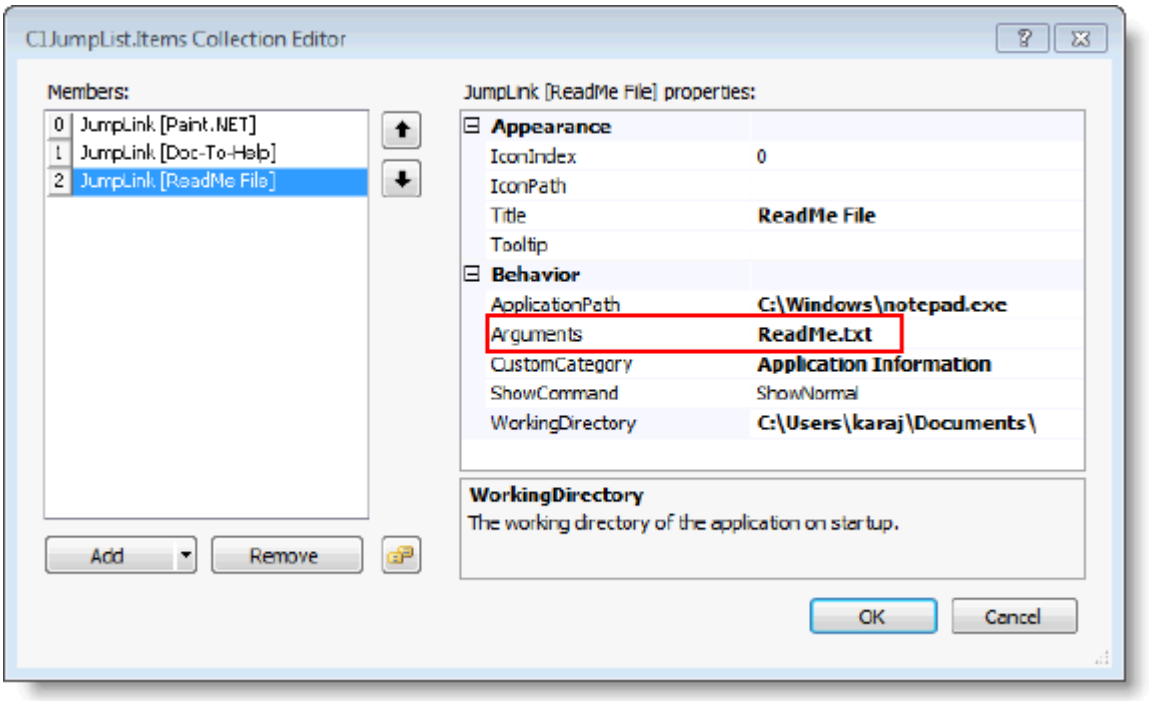
A jump link, represented by the C1JumpLink class and inheriting from the C1JumpItem class, represents a shortcut from the jump list to an application.



To specify the path to the .exe file you wish to link to, just set the ApplicationPath property.

A jump link be easily customized by setting a few properties. You can set the Title property to give your link a title, and you can set the IconPath and IconIndex properties to change the icon associated with the jump link. Without a title, the jump link won't appear in the jump list. However, C1JumpLink will use the application's default .ico file (if it has one) if you don't set the icon properties. The C1JumpLink class also has a CustomCategory property that allows you to group links together under headings.

You can pass arguments to any jump link by setting the Arguments property. For example, you can use the Arguments property to open a specific text file in Notepad, such as in the following example:

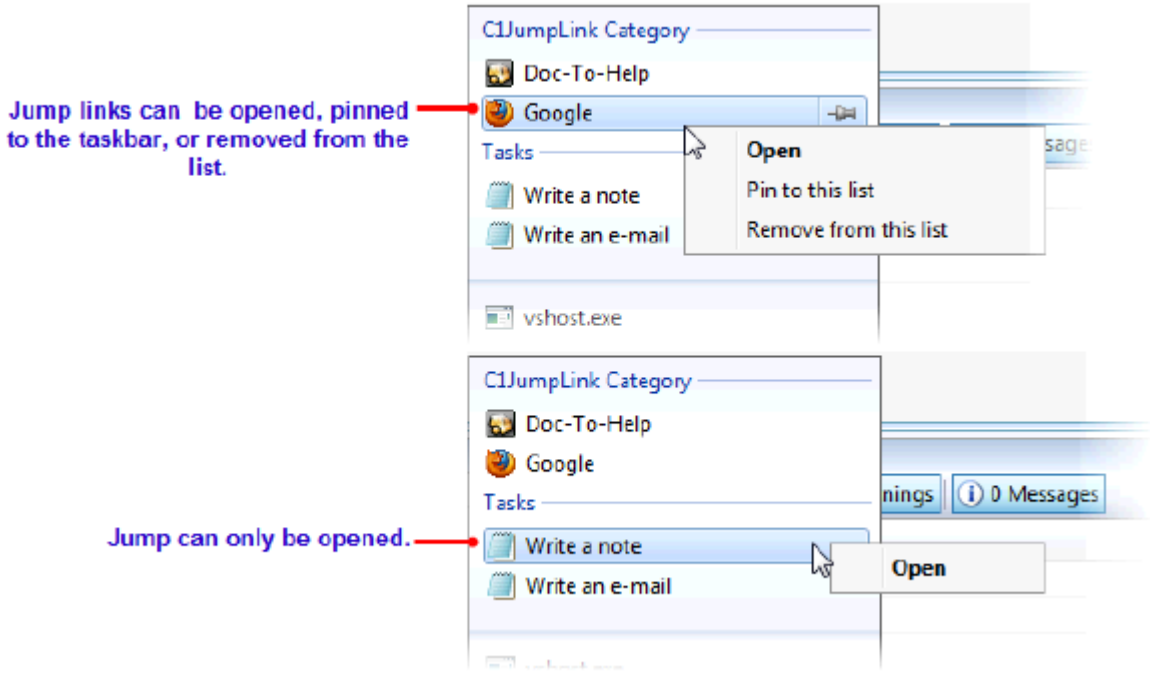


In the above example, the **ReadMe.txt file** resides in the working directory, *C:\Users\karaj\Documents*. If you don't specify your own directory via the WorkingDirectory property, the working directory is assumed to be the directory that the running application resides in.

C1JumpTask Basics

A jump task, which is represented by the C1JumpTask class, is a shortcut to an application. Anything labeled as a C1JumpTask will appear underneath the **Tasks** category.

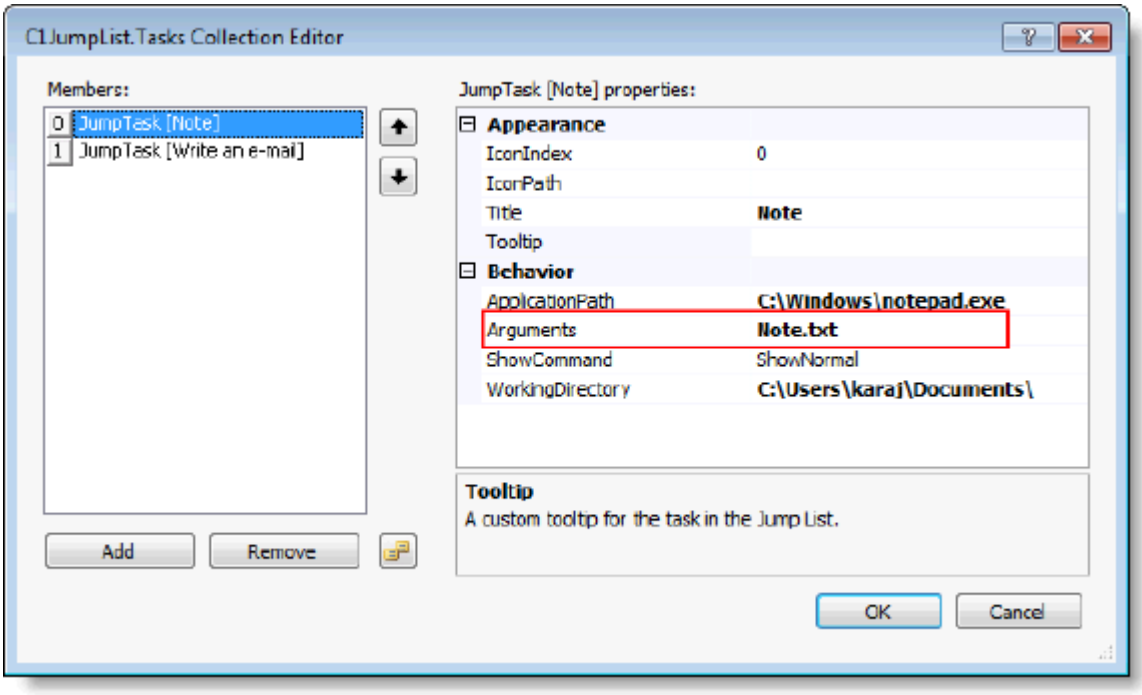
Jump tasks are similar to jump links, only users jump tasks cannot remove jump tasks from the jump list, nor can they pin a jump task to the taskbar.



To specify the path to the .exe file you wish to link to, just set the ApplicationPath property.

A jump task can be easily customized by setting a few properties. You can set the Title property to give your link a title, and you can set the IconPath and IconIndex properties to change the icon associated with the jump task. Without a title, the jump task won't appear in the jump list. However, C1JumpLink will use the application's default .ico file (if it has one) if you don't set the icon properties. C1JumpLink also has a CustomCategory property that allows you to group links together under headings.

You can pass arguments to any jump task by setting the Arguments property. For example, you can use the Arguments property to open a file in Notepad, such as in the following example:



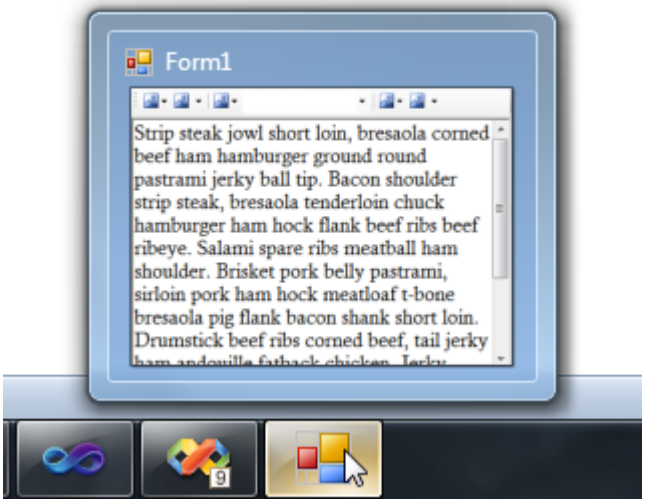
In the above example, the Note.txt file resides in the working directory, *C:\Users\karaj\Documents*. If you don't specify your own directory via the WorkingDirectory property, the working directory is assumed to be the directory that the running application resides in.

Thumbnail Elements

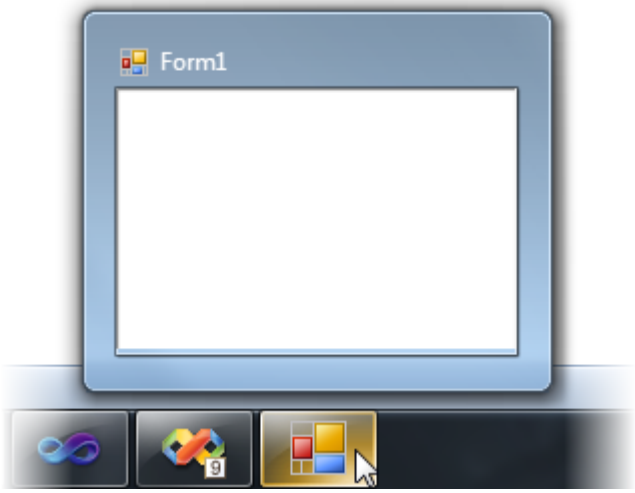
Taskbar button thumbnails, which are represented by the C1Thumbnail class, can consist of two elements: a live preview of the application and a toolbar.

Thumbnail Preview

As long as the user has the thumbnail preview feature activated in Windows 7, a live preview of the entire application window will appear in the thumbnail preview.



If you want users to only see a portion of the application, you can restrict the preview to a portion of the application by setting the ClipControl property. For example, you could restrict the preview to display only an application's rich text editor by setting the ClipControl property to the ID of a **RichTextBox** control.



Thumbnail Toolbar

One of the most convenient additions to the Windows 7 UI was the thumbnail toolbar. These toolbars have cut down on the need for time-wasting application switching. For example, the Windows Media Player now contains Previous, Next, and Play/Pause buttons so that you can switch your music without cluttering your desktop with another window. In the sample image below, you can see a toolbar at the bottom of the thumbnail.

A **Windows 7 Pack for WinForms** thumbnail button is represented by the C1ThumbButton class.

You can add an icon to a C1ThumbButton via the Icon property. By default, each button will have a button border, but you can turn this off by setting the NoBackground property to **True**.

Windows 7 Control Pack for WinForms Samples

Please be advised that this ComponentOne software tool is accompanied by various sample projects and/or demos, which may make use of other development tools included with the ComponentOne Studio.

Please refer to the pre-installed product samples through the following path:

Documents\ComponentOne Samples\WinForms

Click one of the following links to view a list of **Windows 7 Control Pack for WinForms** samples:

Visual Basic samples

Sample	Description
JumpItemDemo	The sample works with the Recent, Frequent, and custom categories in application's jump list. The sample adds custom destinations to a jump list. In addition, it allows to delete destinations from the "known" and custom categories, register/unregister file association, and get the list of items in the "known" (Recent and Frequent) categories. A list of destinations and common tasks associated with an application is attached to that application's taskbar button (as well as to the equivalent Start menu entry). This is the application's jump list. The jump list is available whether the taskbar button is in a launcher state (the application isn't running) or whether it represents one or more windows.
TaskbarDemo	This sample shows a few taskbar extensions using the **C1TaskbarButton** component. **C1TaskbarButton**works with Windows 7 taskbar extensions. You can adjust the application's thumbnail toolbar, icon overlay, and the progress indicator displayed on the taskbar button. In addition, you can select a particular control on a Form and use its contents as the thumbnail image.
TaskDialogDemo	The sample shows various elements of a task dialog box using the **C1TaskDialog** component. Task dialog box is a powerful replacement of the old message box. It consists of several elements, most of which are optional. The **C1TaskDialog** component gives an ability to create and adjust task dialog boxes without much effort. It works like the common dialog boxes (**OpenFileDialog**, for example). You can set a few properties, then call to the Show() method to display the dialog box at run-time.

C# samples

Sample	Description
JumpItemDemo	The sample works with the Recent, Frequent, and custom categories in application's jump list. The sample adds custom destinations to a jump list. In addition, it allows to delete destinations from the "known" and custom categories, register/unregister file association, get the list of items in the "known" (Recent and Frequent) categories. A list of destinations and common tasks associated with an application is attached to that application's taskbar button (as well as to the equivalent Start menu entry). This is the application's jump list. The jump list is available whether the taskbar button is in a launcher state (the application isn't running) or whether it represents one or more windows.
JumpTaskDemo	This sample adds the Task category to a custom jump list. In this sample, we add dynamic Tasks to a jump list. Typically, tasks are **IShellLink** items with command-line arguments that indicate particular functionality that can be triggered by an

	application. The **C1JumpTask** class encapsulates the shell interface. It's easy to manipulate tasks using the **C1TaskbarButton** component.
TaskbarDemo	This sample shows a few taskbar extensions using the **C1TaskbarButton** component. **C1TaskbarButton**works with Windows 7 taskbar extensions. You can adjust the application's thumbnail toolbar, icon overlay, and the progress indicator displayed on the taskbar button. In addition, you can select a particular control on a Form and use its contents as the thumbnail image.
TaskDialogDemo	The sample shows various elements of a task dialog box using the **C1TaskDialog** component. Task dialog box is a powerful replacement of the old message box. It consists of several elements, most of which are optional. The **C1TaskDialog** component gives an ability to create and adjust task dialog boxes without much effort. It works like the common dialog boxes (**OpenFileDialog**, for example). You can set a few properties, then call to the Show() method to display the dialog box at run-time.

Windows 7 Control Pack for WinForms Task-Based Help

The task-based help assumes that you are familiar with programming in Visual Studio. If you are a novice to the **Windows 7 Control Pack for WinForms** product, please see the Windows 7 Control Pack for WinForms Quick Starts first.

Each topic provides a solution for specific tasks using the **Windows 7 Control Pack for WinForms** product. By following the steps outlined in the help, you will be able to create projects demonstrating a variety of **Windows 7 Control Pack for WinForms** features.

Each task-based help topic also assumes that you have created a new .NET project and, unless otherwise stated, have added a reference to the **C1.Win.C1Win7Pack** namespace and added a **C1TaskDialog** or **C1TaskbarButton** component to the application. For additional information on this topic, see Creating a .NET Project.

C1TaskDialog Task-Based Help

The following topics detail information about how you can customize the appearance and behavior of the **C1TaskDialog** control. If you are a novice to using the **C1TaskDialog** control, you may want to see the C1TaskDialog Quick Start first. Note that each of the following task-based help topics assumes that you have created a new .NET project and, unless otherwise stated, have added a reference to the **C1.Win.C1Win7Pack** namespace and added a **C1TaskDialog** component to the application.

Adding Command Link Buttons

Command links have a clean, lightweight appearance that allows for descriptive labels, and are displayed with either a standard arrow or custom icon, and an optional supplemental explanation. For more information, see the Command Links topic. In this topic, you'll add a command link button to an existing **C1TaskDialog** control.

In the Tasks Menu

Complete the following steps to add a command link button with an access key to an existing **C1TaskDialog** control:

1. On the Form in Design view, click once on the **C1TaskDialog** component to select it.
2. Click the control's smart tag to open the **C1TaskDialog Tasks** menu. For more information about the menu, see the C1TaskDialog Tasks Menu topic.
3. In the **C1TaskDialog Tasks** menu, confirm that the **Use Command Links** check box is checked.
4. In the **C1TaskDialog Tasks** menu, click the **Add Custom Buttons** item. The **C1TaskDialog.CustomButtons Collection Editor** will open.
5. In the **CustomButtons Collection Editor**, click the **Add** button to add a new **CustomButton**.
6. Edit the button's **Note** text in the Properties pane, for example set it to "This button is a custom command link."
7. Set the button's **Text** property in the Properties pane to "Command Link Example".
8. Click OK to close the **CustomButtons Collection Editor**.

In the Properties Window

Complete the following steps to add a command link button with an access key to an existing **C1TaskDialog** control:

1. On the Form in Design view, click once on the **C1TaskDialog** component to select it.
2. Navigate to the Properties window and confirm that the UseCommandLinks property is set to **True**.
3. In the Properties window, click the ellipses button next to the **CustomButtons** item. Alternatively, if the

Commands area beneath the Properties window is displayed, you can select the **Edit Custom Buttons** item. The **C1TaskDialog.CustomButtons Collection Editor** will open.

4. In the **CustomButtons Collection Editor**, click the **Add** button to add a new **CustomButton**.
5. Edit the button's **Note** text in the Properties pane, for example set it to "This button is a custom command link."
6. Set the button's **Text** property in the Properties pane to "Command Link Example".
7. Click OK to close the **CustomButtons Collection Editor**.

In Code

The following code adds a command link button to an existing **C1TaskDialog** control:

To write code in Visual Basic

Visual Basic

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
    Dim C1CustomButton1 As New C1.Win.C1Win7Pack.C1CustomButton
    C1CustomButton1.Text = "Command Link Example"
    C1CustomButton1.Note = "This button is a custom command link."
    C1TaskDialog1.CustomButtons.Add(C1CustomButton1)
End Sub
```

To write code in C#

C#

```
public Form1()
{
    InitializeComponent();
    C1CustomButton c1CustomButton1 = new C1CustomButton();
    c1CustomButton1.Text = "Command Link Example";
    c1CustomButton1.Note = "This button is a custom command link.";
    c1TaskDialog1.CustomButtons.Add(c1CustomButton1);
}
```

When You've Accomplished

In this topic, you've added a command link button to an existing **C1TaskDialog** control.

Setting the Access Key

An access key is an underlined character in the text of a menu, menu item, or the label of a control such as a button. With an access key, the user can "click" a button by pressing the ALT key in combination with the predefined access key. For more information, see the Access Keys topic. In this topic, you'll add a custom button with an access key to an existing **C1TaskDialog** control.

In the Tasks Menu

Complete the following steps to add a command link button with an access key to an existing **C1TaskDialog** control:

1. On the Form in Design view, click once on the **C1TaskDialog** component to select it.
2. Click the control's smart tag to open the **C1TaskDialog Tasks** menu. For more information about the menu, see the C1TaskDialog Tasks Menu topic.
3. In the **C1TaskDialog Tasks** menu, uncheck the **Use Command Links** check box.
4. In the **C1TaskDialog Tasks** menu, click the **Add Custom Buttons** item. The **C1TaskDialog.CustomButtons Collection Editor** will open.
5. In the **CustomButtons Collection Editor**, click the **Add** button to add a new **CustomButton**.
6. Set the button's **Text** property in the Properties pane to "&Access Key Example". The ampersand indicates that the letter after the ampersand should be used as the access key.
7. Click OK to close the **CustomButtons Collection Editor**.

In the Properties Window

Complete the following steps to add a command link button with an access key to an existing **C1TaskDialog** control:

1. On the Form in Design view, click once on the **C1TaskDialog** component to select it.
2. Navigate to the Properties window and set the UseCommandLinks property is set to **False**.
3. In the Properties window, click the ellipses button next to the **CustomButtons** item. Alternatively, if the **Commands** area beneath the Properties window is displayed, you can select the **Edit Custom Buttons** item. The **C1TaskDialog.CustomButtons Collection Editor** will open.
4. In the **CustomButtons Collection Editor**, click the **Add** button to add a new **CustomButton**.
5. Set the button's **Text** property in the Properties pane to "&Access Key Example". The ampersand indicates that the letter after the ampersand should be used as the access key.
6. Click **OK** to close the **CustomButtons Collection Editor**.

In Code

The following code adds a command link button with an access key to an existing **C1TaskDialog** control:

To write code in Visual Basic

Visual Basic

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
    C1TaskDialog1.UseCommandLinks = False
    Dim C1CustomButton1 As New C1.Win.C1Win7Pack.C1CustomButton
    C1CustomButton1.Text = "&Access Key Example"
    C1TaskDialog1.CustomButtons.Add(C1CustomButton1)
End Sub
```

To write code in C#

C#

```
public Form1()
{
    InitializeComponent();
    c1TaskDialog1.UseCommandLinks = false;
    C1CustomButton c1CustomButton1 = new C1CustomButton();
    c1CustomButton1.Text = "&Access Key Example";
    c1TaskDialog1.CustomButtons.Add(c1CustomButton1);
}
```

When You've Accomplished

In this topic, you've added a custom button with an access key to an existing **C1TaskDialog** control. Run your application, and when the **C1TaskDialog** dialog box is open press the ALT + A keys to "click" the button.

Setting the Default Button

To set the default button you can use the SetDefaultButton method to the name of the button. The default button indicates which button is clicked when the **C1TaskDialog** control has focus and the user presses the ENTER key.

In the Properties Window

Complete the following steps to set the default button to an existing button:

1. On the Form in Design view, click once on the **C1TaskDialog** component to select it.
2. Navigate to the Properties window and locate the **DefaultButton** item.
3. Click the drop-down area next to the **DefaultButton** item and select the button you want to be set to the default button.

In Code

The following code sets the default button:

To write code in Visual Basic

Visual Basic

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
    C1TaskDialog1.SetDefaultButton(C1CustomButton1)
End Sub
```

To write code in C#

C#

```
public Form1()
{
    InitializeComponent();
    c1TaskDialog1.SetDefaultButton(c1CustomButton1);
}
```

When You've Accomplished

In this topic, you've set the default button to an existing button. If you run the application, you'll see that the button you indicated is set as the default button – if you open the dialog box and press the ENTER key, the button will be selected.

Adding a Progress Bar

You can easily add a progress bar by using the C1ProgressBar class and members to customize the progress bar. A progress bar can be used to indicate the progress of a lengthy operation. The progress bar consists of an area that is filled from left to right as an operation progresses.

In the Properties Window

Complete the following steps to make the progress bar visible:

1. On the Form in Design view, click once on the **C1TaskDialog** component to select it.
2. Navigate to the Properties window and locate and expand the **ProgressBar** item.
3. Click the drop-down area next to the **State** item and select **Indeterminate**.
4. Click the drop-down area next to the **Visible** item and select **True**.

In Code

The following code makes the progress bar visible:

To write code in Visual Basic

Visual Basic

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
    C1TaskDialog1.ProgressBar.State = TaskDialogProgressBarState.Indeterminate
    C1TaskDialog1.ProgressBar.Visible = True
End Sub
```

To write code in C#

C#

```
public Form1()
{
    InitializeComponent();
    c1TaskDialog1.ProgressBar.State = TaskDialogProgressBarState.Indeterminate;
    c1TaskDialog1.ProgressBar.Visible = true;
}
```

When You've Accomplished

In this topic you've made the progress bar visible in a dialog box and set the progress bar's state.

C1TaskbarButton Task-Based Help

The following topics detail information about how you can customize the appearance and behavior of the **C1TaskDialog** control. If you are a novice to using the **C1TaskbarButton** control, you may want to see the C1Taskbarbutton Quick Start first. Note that each of the following task-based help topics assumes that you have created a new .NET project and, unless otherwise stated, have added a reference to the **C1.Win.C1Win7Pack** namespace and added a **C1TaskbarButton** component to the application.

Working with the Jump List

This section features tasks associated with a C1JumpList, including how to add tasks, links, and path jumps to the jump list. Each topic assumes that you have referenced the **C1.Win.C1Win7Pack** namespace in your project.

Adding Jump Tasks

A jump task is a link from the taskbar's jump list to an application. A jump task can be added to the jump list at design time through a collection editor or dynamically at run time.

In the Designer

Complete the following steps:

1. Open the **C1JumpList.Tasks Collection Editor** in one of the following ways:
   - Click the C1TaskbarButtons smart tag to open the **C1TaskbarButton Tasks Menu** and select **Edit Jump Tasks**.

     OR

   - In the **Properties** window, expand the JumpList node and then, next to Tasks, click the ellipses button.
2. In the **C1JumpList.Tasks Collection Editor**, click **Add** to add a jump task. You can set the task's properties in the properties grid.

In Code

The following code creates one C1JumpTask and adds it to the jump list.

To write code in Visual Basic

Visual Basic

```
Dim c1JumpTask1 As New C1JumpTask()
c1TaskbarButton1.JumpList.Tasks.Add(c1JumpTask1)
```

To write code in C#

C#

```
C1JumpTask c1JumpTask1 = new C1JumpTask();
c1TaskbarButton1.JumpList.Tasks.Add(c1JumpTask1);
```

If you'd like to add a range of C1JumpTasks to the jump list, you can use the AddRange method instead, like this:

To write code in Visual Basic

Visual Basic

```
Dim C1JumpTask1 As New C1JumpTask()
Dim C1JumpTask2 As New C1JumpTask()
C1TaskbarButton1.JumpList.Tasks.AddRange(New C1.Win.C1Win7Pack.C1JumpTask()
{C1JumpTask1, C1JumpTask2})
```

To write code in C#

C#

```
C1JumpTask c1JumpTask1 = new C1JumpTask();
C1JumpTask c1JumpTask2 = new C1JumpTask();
c1TaskbarButton1.JumpList.Tasks.AddRange(new C1.Win.C1Win7Pack.C1JumpTask[]
```

```
{c1JumpTask1, c1JumpTask2});
```

Adding JumpLinks

A jump link is a link from the jump list to an application. Jump links can be added at design time using a collection editor, or they may be added dynamically with code.

In the Designer

Complete the following steps:

1. Open the **C1JumpList.Items Collection Editor** in one of the following ways:
   - Click the C1TaskbarButtons smart tag to open the **C1TaskbarButton Tasks Menu** and select **Edit Jump Items**.

     OR

   - In the **Properties** window, expand the JumpList node and then, next to Items, click the ellipses button.
2. In the **C1JumpList.Items Collection Editor**, open the **Add** drop-down list and select C1JumpLink.
3. Set the properties in the properties grid.

In Code

The following code creates one C1JumpLink and adds it to the jump list:

To write code in Visual Basic

Visual Basic

```
Dim c1JumpLink1 As New C1JumpLink()
c1TaskbarButton1.JumpList.Items.Add(c1JumpLink1)
```

To write code in C#

C#

```
C1JumpLink c1JumpLink1 = new C1JumpLink();
c1TaskbarButton1.JumpList.Items.Add(c1JumpLink1);
```

If you'd like to add a range of C1JumpLinks to the jump list, you can use the AddRange method instead, like this:

To write code in Visual Basic

Visual Basic

```
Dim C1JumpLink1 As New C1JumpLink()
Dim C1JumpLink2 As New C1JumpLink()
C1TaskbarButton1.JumpList.Items.AddRange(New C1.Win.C1Win7Pack.C1JumpLink()
{C1JumpLink1, C1JumpLink2})
```

To write code in C#

C#

```
C1JumpLink C1JumpLink1 = new C1JumpLink();
C1JumpLink C1JumpLink2 = new C1JumpLink();
c1TaskbarButton1.JumpList.Items.AddRange(new C1.Win.C1Win7Pack.C1JumpLink[] {
    C1JumpLink1,
    C1JumpLink2});
```

Adding JumpPaths

Jump paths are links from the jump list to a specified file. Please note that your application has to have the file type of this link registered, otherwise Jump Paths will simply not appear on the jump list.

In the Designer

Complete the following steps:

1. Open the **C1JumpList.Items Collection Editor** in one of the following ways:
   - Click the C1TaskbarButtons smart tag to open the **C1TaskbarButton Tasks Menu** and select **Edit Jump Items**.

     OR

   - In the **Properties** window, expand the JumpList node and then, next to Items, click the ellipses button.
2. In the **C1JumpList.Items Collection Editor**, open the **Add** drop-down list and select C1JumpPath.
3. Set the properties in the properties grid. There are only two properties, CustomCategory and Path, available for a C1JumpPath.

In Code

The following code creates one C1JumpPath and adds it to the jump list:

To write code in Visual Basic

Visual Basic

```
Dim C1JumpPath1 As New C1JumpPath()
C1TaskbarButton1.JumpList.Items.Add(C1JumpPath1)
```

To write code in C#

C#

```
C1JumpPath c1JumpPath1 = new C1JumpPath();
c1TaskbarButton1.JumpList.Items.Add(c1JumpPath1);
```

If you'd like to add a range of C1JumpPaths to the jump list, you can use the AddRange method instead, like this:

To write code in Visual Basic

Visual Basic

```
Dim C1JumpPath1 As New C1JumpPath()
Dim C1JumpPath2 As New C1JumpPath()
C1TaskbarButton1.JumpList.Items.AddRange(New C1.Win.C1Win7Pack.C1JumpPath()
{C1JumpPath1, C1JumpPath2})
```

To write code in C#

```
C#
C1JumpPath C1JumpPath1 = new C1JumpPath();C1JumpPath C1JumpPath2 = new C1JumpPath();
c1TaskbarButton1.JumpList.Items.AddRange(new C1.Win.C1Win7Pack.C1JumpPath[] {
    C1JumpPath1,
    C1JumpPath2});
```

Working with the Thumbnail Elements

This section contains task-based help regarding C1Thumbnails. Each topic assumes that you have referenced the **C1.Win.C1Win7Pack** namespace in your project.

Adding Thumbnail Buttons

A C1TaskbarButton control will provide your application with a thumbnail that appears as the user hovers over the application's icon with his or her cursor. This thumbnail window can also contain buttons, such as a next/previous button, that allow users to interact with the application right from the Windows 7 toolbar. These buttons can be added at design time via a collection editor or at run time via code.

In the Designer

1. Open the **C1Thumbnails.Buttons Collection Editor** in one of the following ways:

· Click the C1TaskbarButtons smart tag to open the **C1TaskbarButton Tasks Menu** and select **Edit** Thumbnail Buttons.

OR

· In the **Properties** window, expand the Thumbnail node and then, next to Items, click the ellipses button.

2. In the **C1Thumbnails.Buttons Collection Editor**, click the **Add** button to add a button to the thumbnail.

3. Set the properties in the properties grid.

In Code

To add one button to the thumbnail, create the C1ThumbButton and add it to the thumbnail via the Add method, such as in the following example:

To write code in Visual Basic

```
Visual Basic
Dim C1ThumbButton As New C1ThumbButton()
C1TaskbarButton1.Thumbnail.Buttons.Add(C1ThumbButton)
```

To write code in C#

```
C#
C1ThumbButton c1ThumbButton = new C1ThumbButton();
c1TaskbarButton1.Thumbnail.Buttons.Add(c1ThumbButton);
```

To add two or more buttons to the thumbnail, create the C1ThumbButtons and add them to the thumbnail via the AddRange method, such as in the following example:

To write code in Visual Basic

Visual Basic

```
Dim C1ThumbButton1 As New C1ThumbButton()
Dim C1ThumbButton2 As New C1ThumbButton()
C1TaskbarButton1.Thumbnail.Buttons.AddRange(New C1.Win.C1Win7Pack.C1ThumbButton()
{C1ThumbButton1, C1ThumbButton2})
```

To write code in C#

C#

```
C1ThumbButton C1ThumbButton1 = new C1ThumbButton();
C1ThumbButton C1ThumbButton2 = new C1ThumbButton();
c1TaskbarButton1.Thumbnail.Buttons.AddRange(new C1.Win.C1Win7Pack.C1ThumbButton[]
{C1ThumbButton1,C1ThumbButton2});
```

Restricting the Thumbnail Preview

You can restrict the portion of a window's client area you want to display in the thumbnail preview by setting the ClipControl property to the **ID** of one of the form's elements.

In the Designer

1. Click the smart tag to open the **C1TaskbarButton Tasks** list.
2. Click the **Clip Control** drop-down arrow and select one of the elements on your page from the list. For example, if you had a generic **Button** ("button1") control on your form and wanted to clip the thumbnail image to that button, you'd select button1.

In Code

To restrict the thumbnail image to one element of your form, set the ClipControl property to the **ID** of that element. For example:

To write code in Visual Basic

Visual Basic

```
C1TaskbarButton1.Thumbnail.ClipControl = button1
```

To write code in C#

C#

```
c1TaskbarButton1.Thumbnail.ClipControl = button1;
```

Working with the Taskbar Button

This section contains task-based help regarding the C1TaskbarButton. Each topic assumes that you have referenced the **C1.Win.C1Win7Pack** namespace in your project.

Adding a Progress Indicator to the Taskbar Button

This topic illustrates how to use the C1TaskbarButton control's progress indicator. In this tutorial, you'll set the value of the progress indicator to the value of a track bar.

Complete the following steps:

1. Add a C1TaskbarButton control and a **TrackBar** control to your form.
2. In the Properties window, set the **TrackBar** control's **Maximum** property to "100" so that its maximum value is the same as the progress indicator's default value.
3. With the **TrackBar** control still selected in the Properties window, double-click the **Events** button and then double-click the Scroll event to add a **Scroll** event handler to Code view.
4. Add the following code to the **trackBar1_Scroll** event handler:

   To write code in Visual Basic

   Visual Basic

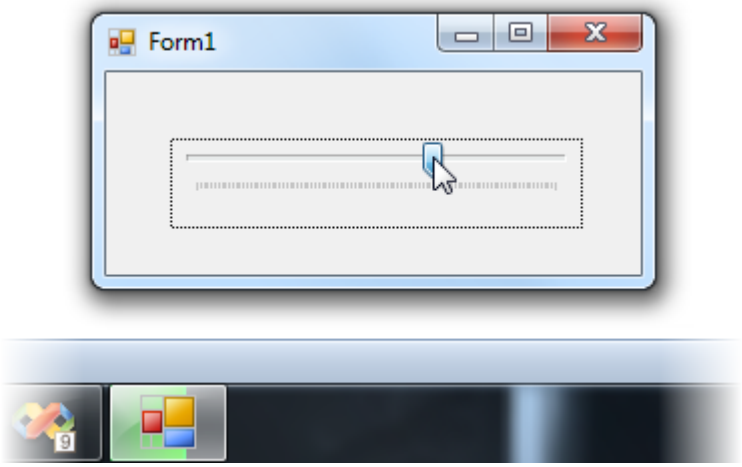
   ```
   Private Sub trackBar1_Scroll(sender As Object, e As EventArgs)
      C1TaskbarButton1.ProgressIndicator.Value = trackBar1.Value
      C1TaskbarButton1.ProgressIndicator.Show()
   End Sub
   ```

   To write code in C#

   C#

   ```
   private void trackBar1_Scroll(object sender, EventArgs e)
      {
          c1TaskbarButton1.ProgressIndicator.Value = trackBar1.Value;
          c1TaskbarButton1.ProgressIndicator.Show();
      }
   ```

5. Press F5 to run the application and slide the track bar. Observe that the progress indicator on the taskbar button advances as you slide the track bar.



Adding an Overlay Image to the Taskbar Button

The overlay image is used to communicate information with a user about the state of an application. The C1TaskbarButtons overly icon can be set at design time in the **Properties** window or can be set via code for run time interaction. It is more common to see this property set when a specific event occurs at run time, but this topic will explain how to do it both ways.

Note: The icon overlay should always be 16 pixels by 16 pixels.

In the Designer

In the Properties window, click the ellipses button next to the OverlayIcon and select your image from the **Image** dialog box.

In Code

Usually, you'll want to use an overlay icon in response to an application state, response, or event. The easiest thing to do is to add your preferred .ico file to the project as a resource or to use the existing system icons.

This example illustrates how to add an overlay icon to a simple **Button_Click** event using system icons.

To write code in Visual Basic

```
Visual Basic
taskbarButton.OverlayIcon = New Icon(SystemIcons.[Error], 16, 16)
```

To write code in C#

```
C#
taskbarButton.OverlayIcon = new Icon(SystemIcons.Error, 16, 16);
```

This example illustrates how to add an overlay icon to a simple **Button_Click** event using an embedded .ico resource named "Error".

To write code in Visual Basic

```
Visual Basic
taskbarButton.OverlayIcon = Properties.Resources.[Error]
```

To write code in C#

```
C#
taskbarButton.OverlayIcon = Properties.Resources.Error;
```