Copyright

Contents

Getting Started

Guided Tours

Control Descriptions

Control Reference

Appendixes

Copyright Notice
Credits

Designer Widgets™

Version 1.00a

This help file was produced using *Doc-To-Help®*, by WexTech Systems, Inc.

Sheridan Software Systems, Inc.
35 Pinelawn Road
Melville, NY   11747
Voice   (516) 753-0985
Fax   (516) 753-3661
BBS   (516) 753-5452

Dedicated to all the hard working people on the Designer Widgets team.

Designed, Developed and Tested By:
Jim Tyminski
Joe Modica
Joe Dour
Bill Batik
Debbie Kufner

Special thanks to: Anne, Rosanne, Patty, Jason, Melissa, Michael, Ashley, Jenna & Nicole

What is Designer Widgets?

Designer Widgets is a set of custom controls that lets you easily add 'state of the art' functionality along with a customizable look to you forms. Each control has all the power and ease of use you have come to expect from Visual Basic controls.

There are 3 controls that make up Designer Widgets. They are:

Related Topics:

Dockable Toolbar
Index Tab
FormFX

Dockable Toolbar

The Dockable Toolbar allows you to easily create and maintain toolbars on your forms. The following is a list of some of its features:

- There can be multiple groups of buttons on each toolbar.
- There can be multiple toolbars on each form.
- Spacing between buttons and groups of buttons is adjustable.
- Toolbars can have a 3-D look.
- Specify separate colors for bevel shadow, highlight and face.
- To minimize system overhead, toolbar buttons are "graphical". In addition, there is a single bitmap that contains all of the button images.
- A mnemonic (accelerator) character can be specified for each button.
- The Toolbar Designer lets you create and maintain toolbars at design time.
- Balloon help is automatically displayed when the mouse pauses over a button for a preset period. (This behavior is optional).

The following features apply to toolbars on MDI forms only:

- Toolbars to be docked to the top, left, right or bottom of the MDI form. They can also be dragged off a docking area and become free floating toolbars.
- Floating toolbars are not restricted to the MDI frame area.
- Floating toolbars can be resized by the user in button increments (the position of each button is adjusted automatically).
- Double clicking on a toolbar toggles its state between floating and docked.

Index Tab

The Index Tab control lets you provide the same 'state of the art' interface that many of today's commercial applications provide. This interface is based on a metaphor that is familiar to all users, that of a series of index cards. The following is a list of some of its features:

- Each index card acts as a separate container allowing you to visually place controls on a tab in design mode as they will appear at runtime.
- There can be multiple rows of index tabs.
- Configurable tab shapes and sizes.
- Each tab can have a caption and/or picture.
- Each tab can be individually enabled/disabled.
- Each tab can be individually made visible/invisible.
- Index tabs can have either a 3D or 2D appearance.
- Specify the font for the active tab.
- Specify a separate picture for the active tab.
- Specify separate foreground and background colors for each tab.
- Specify separate foreground and background colors for the active tab.
- Specify separate colors for bevel shadow, highlight and face.
- Tabs can be activated with the keyboard using mnemonic characters.

FormFX

The FormFX control allows customizing of the caption and client areas of a non MDI form. In addition, FormFX allows you to control the behavior of the form at runtime by restricting size and movement. The following is a list features:

- Adds a 3D look to the caption, border and/or client area.
- Specify separate colors for bevel shadow, highlight and face.
- Add a bitmap, icon or metafile to the caption area.
- Supports multi-line text in captions.
- Includes compete picture and text alignment options for the caption area.
- Allows custom fonts and font sizes for the caption.
- Caption height is modifiable.
- Can automatically adjust size of caption area based on font size.
- Customize the control box, minimize and maximize buttons.
- "InstantClose" feature allows you to close a form with a single click on the control box.
- Lock movement and/or size of form.
- Forms can be kept on top so they never go behind other windows.
- Custom colors for active and inactive windows.

Installing Designer Widgets

This chapter explains how to install Designer Widgets on your computer using the program SETUP.EXE.

Related Topics:

Before You Run Setup
Running Setup
What was installed?

Before You Run Setup

Please take a few minutes before you install Designer Widgets to do the following:

Read the README.TXT File

If there are corrections or additions to this book, they will be listed in a file called README.TXT. This file can be displayed directly from the installation diskette using the Windows™ Notepad utility. After the installation this file can be read by double-clicking the Designer Widgets ReadMe icon in the Program Manager.

Running Setup

When you run the Setup program (SETUP.EXE) to install Designer Widgets on your computer, you'll specify a path for Designer Widgets.

You can start Setup from Windows, or from DOS.

To start Setup from Windows:

1. Insert the Designer Widgets distribution diskette in drive A (or B).
2. In the File Manager, choose Run from the File menu.
3. Type **a:setup** (or **b:setup**).
4. Follow the Setup instructions on the screen.

To start Setup from DOS:

1. Insert the Designer Widgets distribution diskette in drive A (or B).
2. Switch to that drive by **typing a:** (or **b:**)
3. Type **win setup**
4. Follow the Setup instructions on the screen.

What was installed?

The install program will copy all the VBX files into your \WINDOWS\SYSTEM directory (optional). All other files will be located in the directory that was specified during setup. For a description of these files refer to the 'Included Files' section of Appendix A.

If the file VBRUN300.DLL is not already in your Windows System directory, the install program will copy it there. This file is needed by the Toolbar Designer.

Using Designer Widgets

This chapter explains how to use the Designer Widgets custom controls in the Visual Basic environment.

Related Topics:

Including Designer Widgets in Your Project

Including Designer Widgets in Your Project

In Visual Basic, custom controls are installed on a project basis.   However, once you have included a custom control in a project and saved the project, the control will appear in the Toolbox whenever you subsequently open the project.

To get the Designer Widgets icons to appear in your Visual Basic Toolbox, first open the project in which you want to use the Designer Widgets.   Then use the 'Add File' option on the 'File' menu to add the VBX file to your project.   Depending upon which custom control you want to use in your project, you will need to load a different VBX file as follows:

SSDOCKTB.VBX	SSToolbar
SSIDXTAB.VBX	SSIndexTab
SSFORMFX.VBX	SSFormFX

The icons for the Designer Widgets custom controls you selected will then appear in your Toolbox.

Refer to the *'Understanding Projects'* and *'Editing the AUTOLOAD.MAK File'* section in Chapter 5 of the *Visual Basic Programmer's Guide* for a detailed explanation of the above process.

Once the controls are loaded in the Toolbox, you can use them just like any standard VB control.   Also, there are no separate design time and runtime versions of the controls.   The same VBX files you develop with can be shipped with your application.

Using Help

Online Help is a comprehensive reference for nearly all aspects of Designer Widgets. Most of the information contained in this manual is also contained in the online Help.

Related Topics:

4 Ways to Get Help

4 Ways to Get Help

Online Help for Designer Widgets can be invoked by any of the following methods:

1. Double-click the Designer Widgets Help icon in the Program Manager. This will bring you into the main index for help on Designer Widgets.
2. Select a control on the Visual Basic toolbox and press F1. This will bring you into help for the currently selected control. You can then go to the index or search for help on other topics.
3. Clicking of F1 from the property window will give you help on the selected property.
4. When the Procedure box in the VB Code window has the focus, pressing F1 will bring you into help for the highlighted event.

Using the Dockable Toolbar

This chapter guides you through the creation of some sample programs using a Designer Widgets control called the Dockable Toolbar. For a complete description of this control refer to Chapter 8 - 'The Dockable Toolbar'.

Related Topics:

Exercise 1: Creating a Toolbar on an MDI Form
Exercise 2: Creating a Second Toolbar
Exercise 3: Providing Status Bar Help
Exercise 4: Putting Code Behind the Buttons
Exercise 5: Creating a Stationary Toolbar

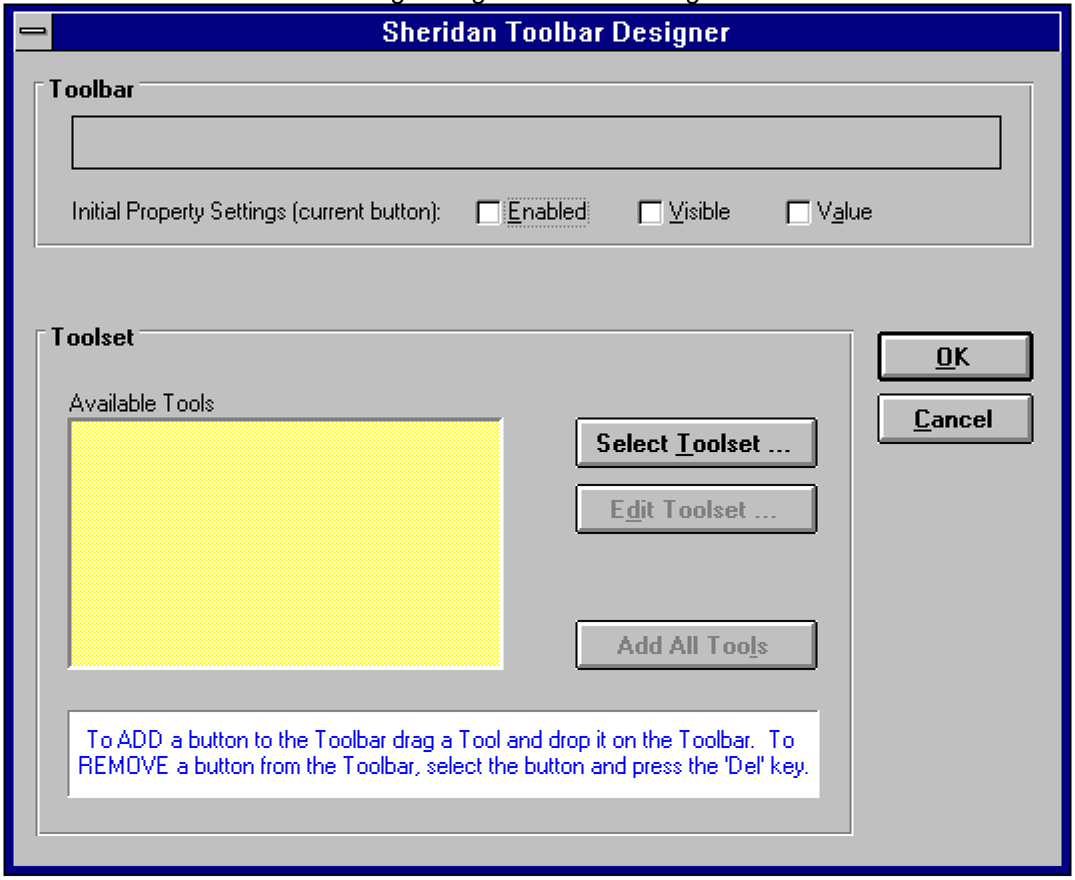
Exercise 1: Creating a Toolbar on an MDI Form

In this exercise we will add a single toolbar to an MDI form.

1. First, run Visual Basic, start a new project and add the SSDOCKTB.VBX file (see Chapter 3 - 'Using Designer Widgets').
2. Next create an MDI form (using the 'File', 'New MDI Form' menu option).
3. Place an SSToolbar control directly on the MDI form by double clicking on the tool in the Visual Basic toolbox.
4. In the Visual Basic properties window set the following properties:

   FloatingCaption = 'Standard'

   Name = 'Tb1'
5. Double click on the **(ToolbarDesigner)** property. This will invoke the Toolbar Designer. For a complete explanation of the Toolbar Designer refer to the Chapter 8 - 'The Dockable Toolbar' under the section heading 'Using the Toolbar Designer'.

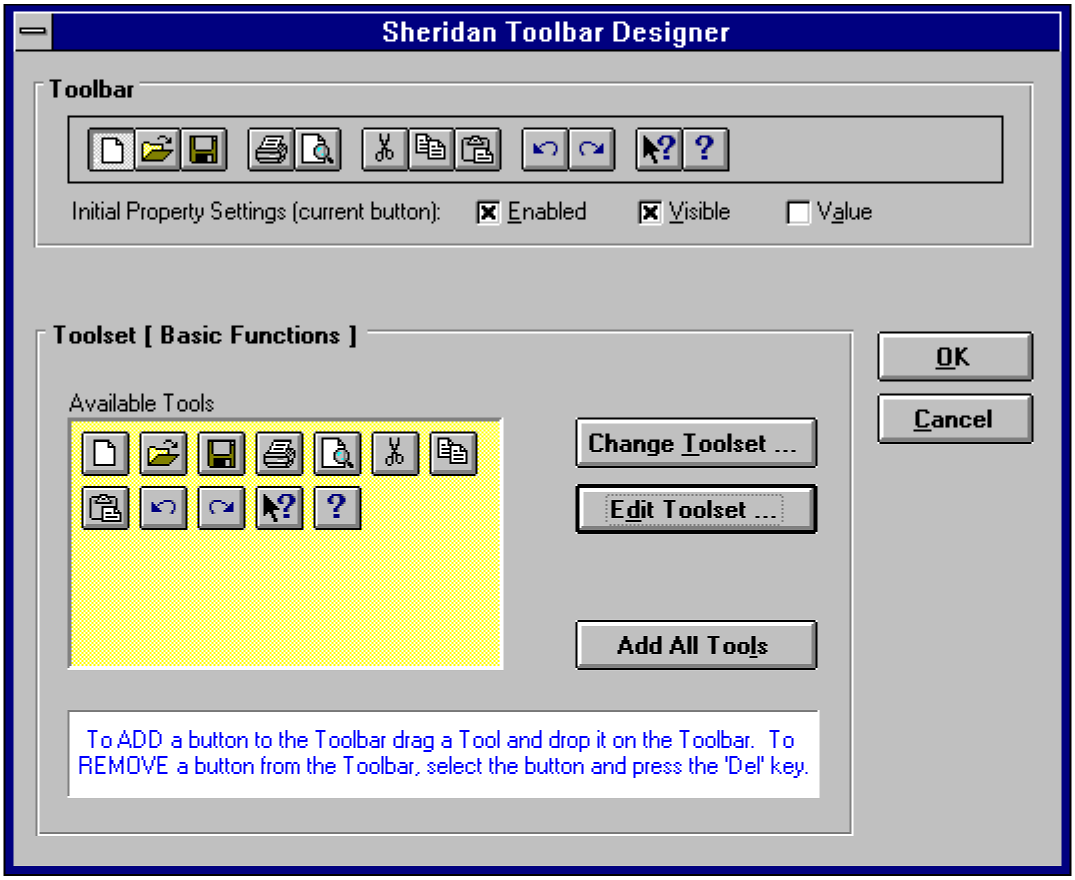


Initial Property Settings (current button): □ Enabled □ Visible □ Value

To ADD a button to the Toolbar drag a Tool and drop it on the Toolbar. To REMOVE a button from the Toolbar, select the button and press the 'Del' key.

6. Next press the 'Select Toolset...' button. This will bring up a file open dialog. Select the file named TSBASIC.BMP in the TOOLSETS sub directory provided with Designer Widgets.
7. Now drag each tool (one at a time) from the 'Available Tools' box into the rectangle labeled 'Toolbar' or press the button labeled 'Add All Tools'.

   The Toolbar Designer should now look like this:



Initial Property Settings (current button): ☒ Enabled ☒ Visible □ Value

To ADD a button to the Toolbar drag a Tool and drop it on the Toolbar. To REMOVE a button from the Toolbar, select the button and press the 'Del' key.

8. Now you can press the 'OK' button.
9. Bring up the Visual Basic 'Project Options' dialog (via the 'Options', 'Project...' menu) and set this MDI form as the 'Start Up Form'.
10. At this point you should save the project.
11. Now we can run the project by pressing the F5 key. The toolbar should be in the top docking area.
12. Click on an area of the toolbar where there is no button and drag the toolbar off the top docking area. The toolbar should now be on a floating window with a small caption of 'Standard'.
13. Try resizing the floating toolbar with the mouse. It should resize itself in button increments and adjust the location of the buttons automatically.
14. Now drag it over to the left docking area. As you move over this area the outline should change to a vertical orientation. When it does, release the mouse. Try dragging and dropping the toolbar onto the bottom and right docking areas as well.
15. Now double click on the toolbar (not on one of its buttons). The toolbar should leave the docking area and become floating. Double click on it again and it will return to its previous docking location.

Exercise 2: Creating a Second Toolbar

Creating a second toolbar is easy. Just place another SSToolbar control on the same MDI form that we created in exercise 1.

1. In the Visual Basic properties window set the following properties:

   FloatingCaption = 'Format'

   Name = 'Tb2'

2. Double click on the **(ToolbarDesigner)** property.
3. Press the 'Select Toolset...' button but this time select the file named TSFORMAT.BMP in TOOLSETS.
4. Again, drag each tool from 'Available Tools' onto the toolbar.
5. Press the 'OK' button.
6. At this point you should save the project.
7. Run the project. Notice that the two toolbars can share docking areas or float independently.

Note *Both toolsets used in examples 1 and 2 contain button images that are the same size (24 pixels wide by 22 pixels high) and none of the buttons contain any text. However, you can make the button images larger or smaller as long as all images within a single toolset are of equal size.*

If you want to include text in the buttons you will need to make a bitmap with larger buttons and place the text in the bitmap. This can be done with a bitmap editor like the standard Paint Brush application (PBRUSH.EXE). Refer to 'Terms and Concepts' in Chapter 8 - 'The Dockable Toolbar' for a complete discussion of toolset bitmaps.

*You can even specify an accelerator character for each tool via the **ToolsetToolMnemonic()** property. However, you should be aware that accelerator characters only work within the current active form. Therefore, if a MDI child form is active, the accelerator characters for toolbars that are on the MDI parent will not operate.*

Also note that if you have more than one toolbar on an MDI form and the sizes of the buttons in one toolbar are not the same size as the buttons in another toolbar then the docking area sizes will be computed based on the largest button on any toolbar on the form. This includes toolbars that are not visible.

Exercise 3: Providing Status Bar Help

This exercise assumes you have the THREED.VBX supplied with the Visual Basic Professional Edition. If not, you can create a 2D status bar by substituting a picturebox and a label for the 2 Panel controls.

1. Add THREED.VBX to the project that you have saved in exercises 1 and 2.

2. On the MDI form place a panel control using the ▣ tool.
3. Set its properties as follows:
Align = '2 - Align Bottom'
Caption = ''
BevelOuter = '2 - Raised'
BevelInner = '0 - None'
BevelWidth = 1
4. Now carefully draw another panel on top of the first so that it is a few pixels shorter than the first panel.

5. Set its properties as follows:

   Align = '0 - None'

   Caption = ''

   Name = 'PnStatus'

   BevelOuter = '1 - Inset'

   BevelInner = '0 - None'

   BevelWidth = 1

6. Now position this child panel so that it looks like a section of a status bar.

7. In the MouseEnter event of both SSToolbar controls place the following code (where Tbx is the name of the toolbar):

```
Sub Tbx_MouseEnter(ToolID As String, ToolNum As Integer, Btn As Integer)
pnStatus.Caption = TBx.ToolsetToolDesc(ToolNum)
End Sub
```

8. In the MouseExit event of both SSToolbar controls place the following code:

```
Sub Tbx_MouseExit(ToolID As String, ToolNum As Integer, Btn As Integer)
pnStatus.Caption = ""
End Sub
```

9. At this point you should save the project.

10. Run the project and move the cursor over each button. As you do you should see the status bar display a description of each button's function.

You may have already noticed that when you pause the cursor over a button in one of the toolbars, balloon help is displayed. The help text for each tool is accessed via the **ToolsetToolHelp()** property. This behavior can be controlled with the **BalloonHelp** and **BalloonHelpDelay** properties. Also, if you wanted to delay the display of the status bar help to coincide with balloon help (after the delay interval has expired) then in step 7 above instead of placing the code in the **MouseEnter** event move it to the **Help** event.

Note *All the properties that relate to toolsets including **ToolsetToolHelp()** and **ToolsetToolDesc()** can be set at design time using the Toolbar Designer (see 'Using the Toolbar Designer' in Chapter 8 - 'The Dockable Toolbar'). However, all the toolset properties are also settable from code (see 'Creating a Toolbar at Runtime' in Chapter 8 - 'The Dockable Toolbar').*

Exercise 4: Putting Code Behind the Buttons

Now we need to put some code behind the buttons on the toolbars. Although the following code doesn't do anything useful it demonstrates where the actual code should go.

1. In the Click event of the toolbar that we created in exercise 1 place the following code:

```
Sub Tb1_Click (ToolID As String, ToolNum As Integer,  Btn As Integer, Value As
Integer)

    Select Case ToolID
        Case "ID_NEW"
            MsgBox "This invokes the New File function"
        Case "ID_OPEN"
            MsgBox "This invokes the Open File function"
        Case "ID_SAVE"
            MsgBox "This invokes the Save File function"
        Case "ID_PRINT"
            MsgBox "This invokes the Print function"
        Case "ID_PREVIEW"
            MsgBox "This invokes the Print Preview function"
        Case "ID_CUT"
            MsgBox "This invokes the Cut function"
        Case "ID_COPY"
            MsgBox "This invokes the Copy function"
        Case "ID_PASTE"
            MsgBox "This invokes the Paste function"
        Case "ID_UNDO"
            MsgBox "This invokes the Undo function"
        Case "ID_REDO"
            MsgBox "This invokes the Redo function"
        Case "ID_HELP1"
            MsgBox "This invokes the Item Help function"
        Case "ID_HELP2"
            MsgBox "This invokes the General Help function"
    End Select

End Sub
```

2. At this point you should save the project.
3. Run the project and click on any of the buttons in the toolbar. The appropriate message should be displayed.
4. If you want to put code behind the toolbar created in exercise 2, the **ToolsetToolID()** property settings are "ID_BOLD", "ID_ITALIC", "ID_UNDERLINE", "ID_LEFT", "ID_CENTER", "ID_RIGHT" and "ID_JUSTIFY" respectively.

Exercise 5: Creating a Stationary Toolbar

In this exercise we will add a stationary toolbar to an MDI form. The toolbar will behave as it would on a standard Visual Basic non-MDI form.

This exercise also assumes you have the THREED.VBX supplied with the Visual Basic Professional Edition. If not, you can substitute a picturebox for the Panel control.

1. Create a new project and add the SSDOCKTB.VBX and THREED.VBX files.
2. Next create an MDI form (using the 'File', 'New MDI Form' menu option).
3. Place a Panel control directly on the MDI form by double clicking on the tool in the Visual Basic toolbox.
4. In the Visual Basic properties window set the following properties:

AutoSize	= '3 - AutoSize Child to Panel'
Caption	= ''
BevelOuter	= '0 - None
BevelInner	= '0 - None
BevelWidth	= 0
BorderWidth	= 0

5. Select the tool in the Visual Basic Toolbox and draw the toolbar on top of the panel. Do not double click on the tool to create the control since this will create the toolbar as a child of the form instead of a child of the panel.
6. Now, set the **Outline** property on the toolbar control to **False**.
7. Repeat steps 1 through 7 from exercise 1.
8. Run the project. Now the toolbar is fixed and you won't be able to drag it as you did in exercise 1.

Note *These exercises were intended to give you a feel for the power and flexibility of this control. For a greater understanding of the properties and events associated with the Dockable Toolbar, see Chapter 8 - 'The Dockable Toolbar', Chapter 11 - 'Properties Reference' and Chapter 12 - 'Events Reference'.*

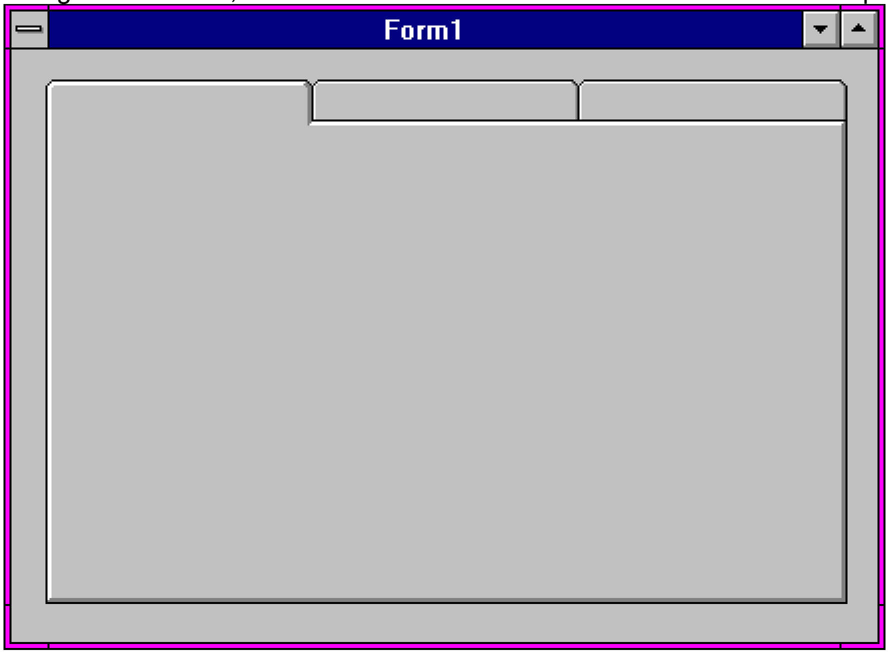
Using the Index Tab

This chapter guides you through the creation of a sample program using a Designer Widgets control called the Index Tab. The Index Tab is a custom control that you can use in your projects to provide the same 'state of the art' interface used by many of today's commercial applications.

The Index Tab control provides an easy way of presenting several dialogs or screens of information on a single form. And since it uses a common metaphor, index tabs, users will feel comfortable with it immediately. When the user clicks on an index tab, the controls associated with that tab will appear and the user can then view or modify the information.

First start a new project and add the SSIDXTAB.VBX file (see Chapter 3 - 'Using Designer Widgets').

Using the tool, draw an Index Tab control on the form so that it occupies most of the form.



Related Topics:

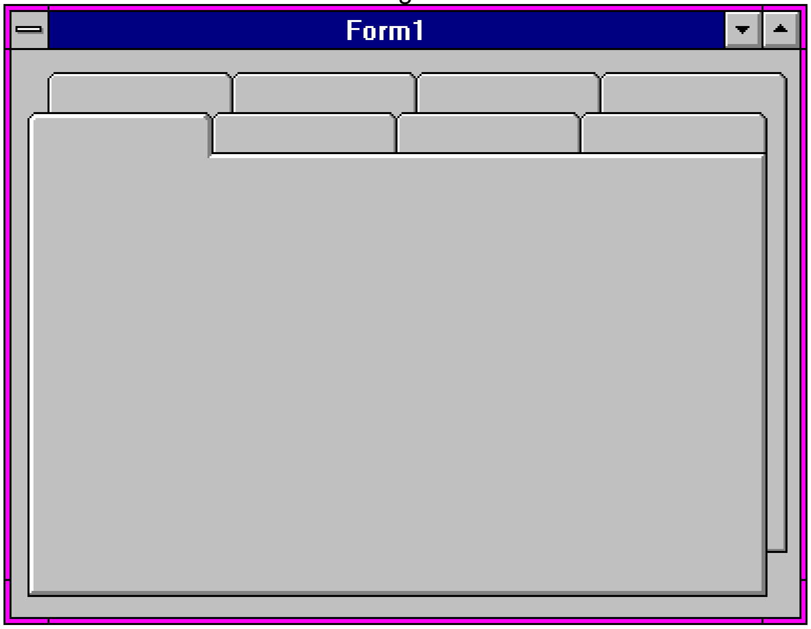
Exercise 1: Setting the Number of Tabs and Rows
Exercise 2: Further Defining the Look of the Tabs
Exercise 3: Adding Captions and Pictures to Tabs
Exercise 4: Changing the Look of the Active Index Tab
Exercise 5: Adding Controls to the Index Tabs at Design time

Exercise 1: Setting the Number of Tabs and Rows

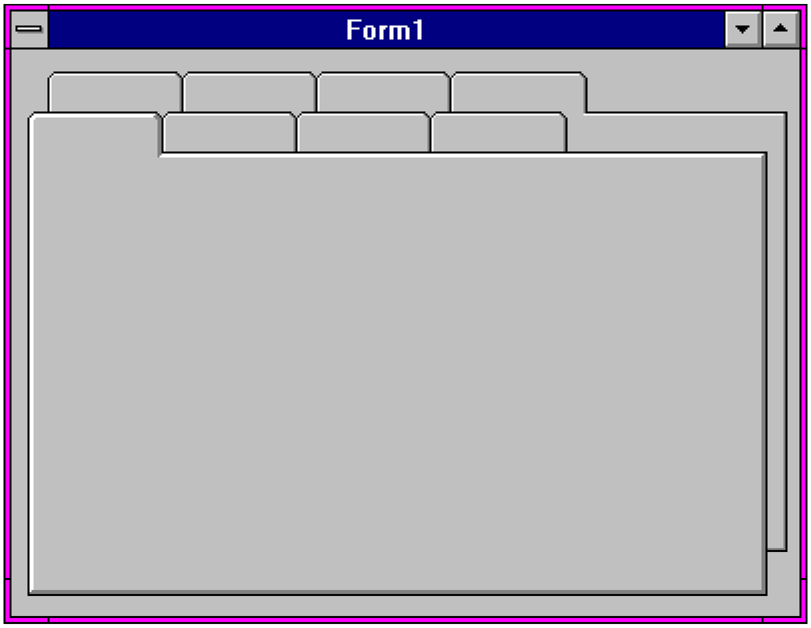
The first step you need to do when using the Index Tab is to setup the basic look of the control. This is usually determined based on the number of tabs you will be using and how wide you need each tab to be to accommodate caption text and/or a picture.

There are three properties that you need to set to get the look you want. These properties are **Tabs**, **TabsPerRow**, and **TabMaxWidth**. When you first create the control on the form, you will notice that the control defaults to a single row of three tabs. Let's use these three properties to change that. Do the following:

1. Set the **Tabs** property to 8. You will notice that 8 tabs will appear and the number of rows has changed to 3.
2. Set the **TabsPerRow** property to 4. You now will see that the number of tabs is still 8, but the number of rows has changed to 2.



3. Notice how the tab widths are all the same and that they are automatically sized to fit exactly on the Index Tab control. To see this better, adjust the width of the control with the mouse.
4. We can override this behavior so that the tab widths can never go over a specific size. This is done with the **TabMaxWidth** property. Setting this to 0 (default) will cause the control to automatically size the tabs. Try setting this to different values to see its effects. These values are in the scale mode of the form or container, which in this case is twips.



These three properties (**Tabs**, **TabsPerRow**, and **TabMaxWidth)** determine the basic look of the Index Tab control.   There are several other properties that change other aspects of the look of the control such as **TabRowOffset**, **TabHeight**, **TabCutSize**, and more.

Exercise 2: Further Defining the Look of the Tabs

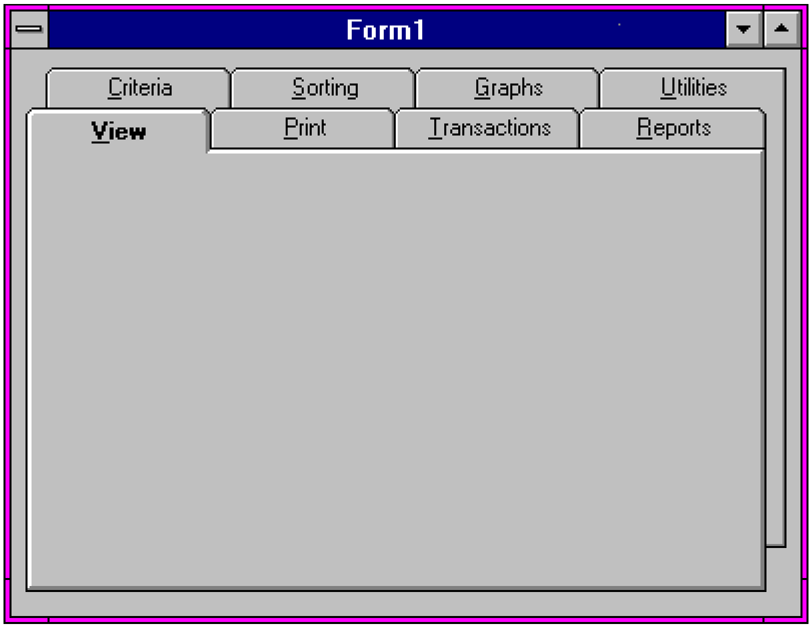
Using the form we created in the previous exercise, we will now walk through the setting of other properties that further define the look of the control.

1. By default, the control sets the height of the tabs to a height that should accommodate a single line of text. However, you may want to specify a larger font or place a picture on each tab. To adjust the height of each tab on the control, set the **TabHeight** property. Try setting this property to different values to see its effects. Remember these values are in the scale mode of the form.
2. The control displays all tabs with angled or cut corners. By default, the size of the cuts are 3 pixels. If you want, you can adjust this size by setting the **TabCutSize** property (in pixels). Experiment with different values to get the angle you want.
3. Notice that the top row of tabs has been indented to the right of the bottom row. If there were three rows of tabs, each row would be indented by the same amount. To adjust the amount of this indent, set the **TabRowOffset** property. This property's value is in the scale mode of its form or container, which in this case is twips.
4. The **TabOrientation** property determines where the tabs are placed. This property defaults to displaying the tabs along the top of the control. However, they can also be shown on the bottom, left or right. Experiment with different settings get the look you want.

Exercise 3: Adding Captions and Pictures to Tabs

The Index Tab control allows you to specify a different caption and/or picture for each tab in the control. Since the **TabCaption()** and **TabPicture()** properties are only available at runtime, the Index Control has provided the **Caption** and **Picture** properties for design time support.

1. First, select a tab by clicking on it with the left mouse button.
2. Now set the **Caption** property to the text you want to appear on that tab.   Remember that if you want mnemonic support, include an ampersand ('&') character before the mnemonic character in the text.



3. If you want to include a picture on that tab, set the **Picture** property as well.
4. Repeat steps 2 and/or 3 for each index tab.
5. Now that you have defined what will appear on each tab, you may want to specify how the information will be aligned. This is done via two properties called **AlignmentCaption** and **AlignmentPicture**. Try different settings on each of these properties to see how they work.

Exercise 4: Changing the Look of the Active Index Tab

When a user selects a tab, that tab becomes the "active tab". When a tab becomes active, all controls associated with that tab will appear. The Index Tab control optionally lets you distinguish the look of the active tab through several properties. With these properties, you can define different fonts, colors or a specific picture to set on the active tab apart.

1. By default, the control will use the default font to display text on the active tab but in bold to distinguish it from the rest. You can adjust the font attributes of the active tab by setting the **ActiveTabFontBold**, **ActiveTabFontItalic**, **ActiveTabFontStrikeThru** and/or **ActiveTabFontUnderline** properties. Experiment with different variations of these properties to get the look you want.

2. You can also set the color of the text in the active tab. This is done via the **ActiveTabForeColor** property.

   Note *Setting the background color is also possible, but in this exercise we are showing the Index Tab in 3D which ignores all background color definitions.*

3. Now to finish the look of the active tab, let's specify a picture to use. Set the **ActiveTabPicture** property to a bitmap that will distinguish the active tab.

Exercise 5: Adding Controls to the Index Tabs at Design time

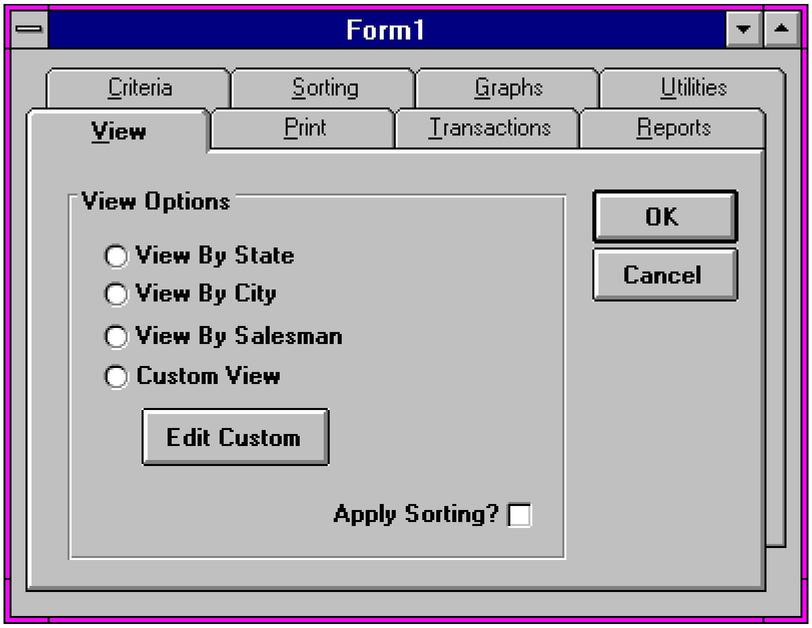
Now that we have the basic look of the form defined, we can show you the most important aspect of the Index Tab control, placing other controls on each index tab. At design time, you can select each tab by clicking on it with the left mouse button. When you select a new tab, the 'client area' (the section of the Index Tab control on which to place other controls) clears giving you a new area to place child controls associated with that tab. When you select another tab, these controls will 'disappear' until you re-select the tab.

1. First, select an index tab by clicking on it with the left mouse button.

2. Draw a frame control on the Index Tab control.

Note *Be sure that the frame control is a child control of the Index Tab. This is how the Index Tab control knows which controls are on which tab. To ensure that the frame is a child of the Index Tab, first select the Index Tab control that you have on the form by clicking on it once with the left mouse button. Then select the frame control from the Visual Basic toolbox. Now drag the frame to the desired size directly on the Index Tab control.*

To test whether the frame is a child of the Index Tab control, move the Index Tab control to a different location on the form. The frame control should follow the Index Tab control. If it does not, try the above steps again. If you have trouble, refer to the Microsoft Visual Basic Programmer's Guide*, Chapter 3* - 'Grouping Options with Option Buttons'; 'Containers for Controls' *on page 46*.



3. Place any other controls that you want to appear on this tab when the user selects it.
4. Repeat the above steps for each tab on the Index Tab control.

5. At design time, you will notice that when you select a new index tab, all the controls associated with the new tab appear and the other controls associated with the previous tab disappear. Click on the different index tabs to see this effect.

6. Once you placed all the controls on the different tabs, run the program. You will see that when you click a tab, only those controls that you placed on that tab will appear.

Congratulations! You have just completed the tour of the Index Tab control. Now that we have illustrated the basic concepts of the Index Tab control, you can use it to enhance your own applications to provide your users a powerful, state-of-the-art interface that is very easy to use.

For a complete description of the Index Tab control and its properties refer to Chapter 9 - 'The Index Tab Control' and Chapter 11 - 'Properties Reference'

Using FormFX

This chapter will walk you through a sample session creating a customized form using the FormFX control. FormFX is a custom control that you can use on your forms to change their appearance, give them a 3D look, adjust the size and contents of their caption areas and/or restrict their size and movement.

Note *At the beginning of most of the following exercises, we will be creating a new project. Each time you start the new project, be sure to add the SSFORMFX.VBX file to your project.*

Related Topics:

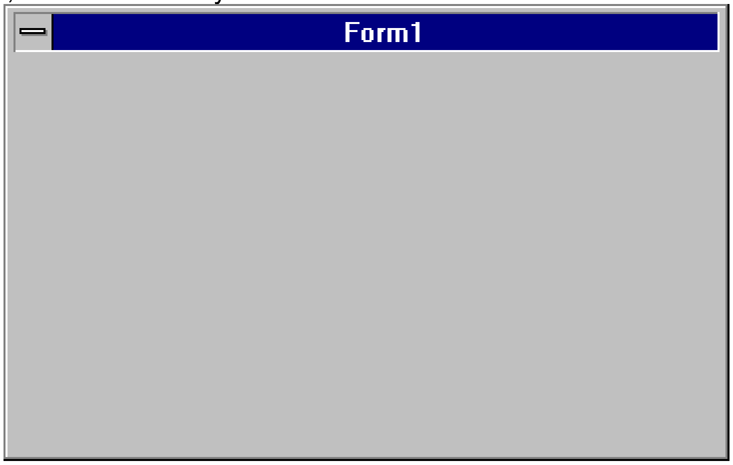
Exercise 1: Creating a 3D Dialog Box
Exercise 2: Adding More 3D Effects to a Form
Exercise 3: Creating a Form with a Small Caption
Exercise 4: Changing the Look of the Caption Buttons
Exercise 5: Custom Caption with Multi-line Text and a Picture
Exercise 6: Changing Runtime Behavior

Exercise 1: Creating a 3D Dialog Box

Using the FormFX control, 3D effects can be applied to several parts of the form such as the caption area, the window border and even the client area. This exercise will show the various properties of the FormFX control that allow you to create a dialog box similar to those found in the latest versions of many of today's commercial applications.

1. Start a new project and open a form.
2. Set the Form's **BorderStyle** property to '3 - Fixed Double', and its **BackColor** property to light gray.
3. Place a single FormFX control anywhere on the form.

4. Set FormFX control's **StdCaption** property and its **Border3D** property to **True**. Now run the program, this is the effect you should have:

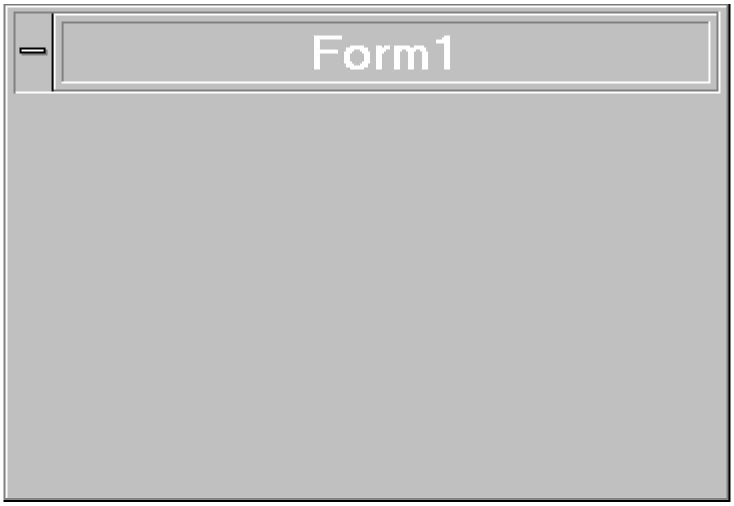


Exercise 2: Adding More 3D Effects to a Form

As we stated in the previous exercise, 3D effects can be applied to several parts of the form using the FormFX. This exercise will show various other properties of the FormFX control that allow you to further define 3D effects on a form.

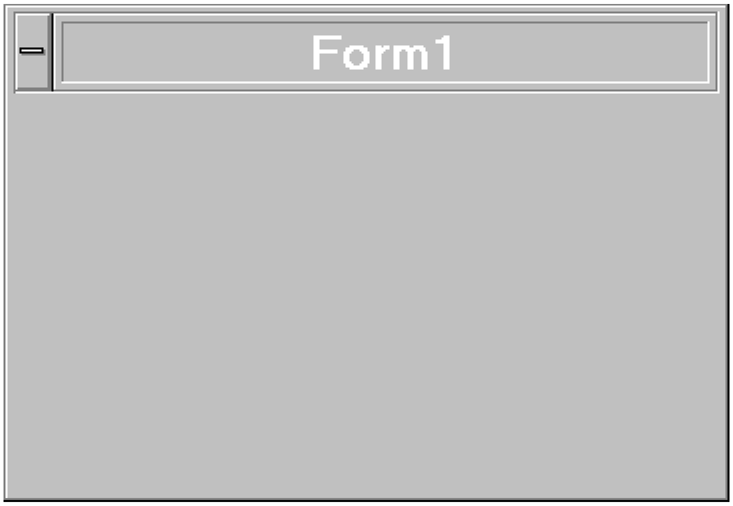
1. Using the same form as in the previous exercise, set the **StdCaption** property of the FormFX control back to **False**. This enables us to customize the caption area.
2. Set the **Caption3D** property to **True**. This will cause the FormFX control to apply beveling to the caption area. To specify the type of beveling to use, set the **CaptionBevelInner** and **CaptionBevelOuter** properties. For this example, set the **CaptionBevelInner** property to 'Inset' and the **CaptionBevelOuter** property to 'Raised'.
3. Now we can adjust the caption so that it is large enough to accommodate the 3D bevels. There are two ways to do this. You can either set the **CaptionHeight** property or let the control adjust the caption automatically based on the font.

   To see the latter method, first set the **CaptionHeight** to zero. Now select a font in the **FontName** property and adjust its size via the **FontSize** property. For this exercise, set the **FontName** to 'MS Sans Serif' and **FontSize** to 18.
4. Now run the program. This is how it should look:



5. Now let's add one more finishing touch. Notice the control box in the caption. Currently, it is only 2D and doesn't seem to fit with the rest of the form. Let's make it 3D by setting the **ControlBox3D** property to **True.**

Now, that's better:

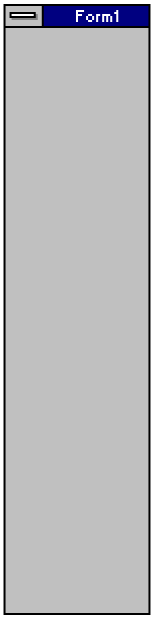


Form1

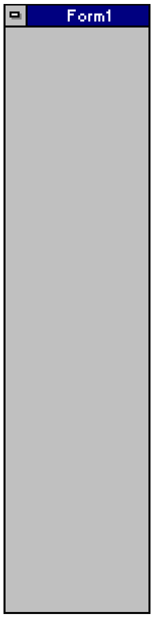
Exercise 3: Creating a Form with a Small Caption

In this exercise we will change the size of the caption to create a small caption (similar to the caption size of the Visual Basic toolbox and that of the custom property dialogs in Sheridan Software's VBAssist).

1. Start with a new form.
2. To follow the style of Visual Basic's toolbox, set the *Form's* **BorderStyle** property to '1 - Fixed Single' and set the **MinButton** and **MaxButton** properties to **False**. Also, size the form so that it has a shape similar to the Visual Basic toolbox
3. Place a FormFX control anywhere on it.
4. Now set FormFX control's **FontName** property to 'Small Fonts' and the **FontSize** property to 6. Be sure that the **CaptionHeight** property is set to zero so that the FormFX control will automatically set the height of the caption area to the size of the font.
5. Now run the program, you should see a form similar to this:



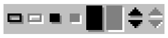
6. This looks fine except for the control box, it's too wide. Well, we can even adjust the width of that. To do this, set the **ControlBoxWidth** property to 10 (this property is specified in pixels). Notice that the previous setting of zero automatically sizes the control box to the standard size. Now, when you run the project the form should look just like the Visual Basic toolbox:


Form1

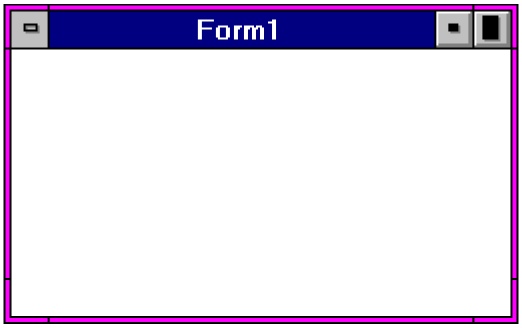
Exercise 4: Changing the Look of the Caption Buttons

So far we have been able to change *several* characteristics of a form using the FormFX control. In this exercise we will actually change the buttons that appear in the caption area.

1. Start with a new form and place the FormFX control anywhere on it.
2. On the FormFX control, set the **CaptionButtonsPic** property to BUTTONS.BMP (this file can be found in the SAMPLES\CHAPTER7 directory). BUTTONS.BMP is a bitmap file that contains the images for all of the caption buttons including the control box and the minimize, maximize and restore buttons. Furthermore, it contains the images for the up and down state of each button. You can create your own bitmap using any bitmap editor such as the standard PaintBrush application (PBRUSH.EXE) but you may want to use this bitmap as a guide. This is what it looks like:



3. Now run the program. As you can see, the caption buttons now look like the ones from the bitmap.



Exercise 5: Custom Caption with Multi-line Text and a Picture

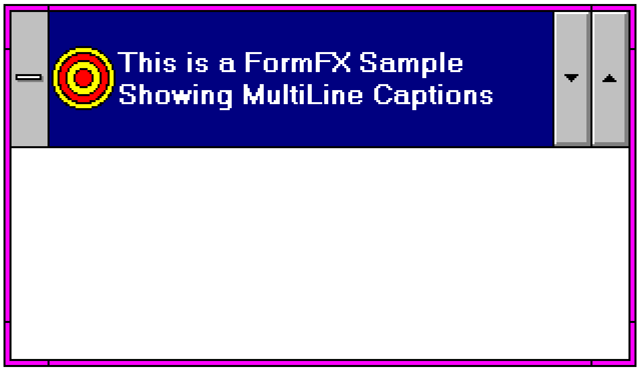
Normally, a form can only have a single line of text centered in its caption area. There is also no way of putting a picture in the caption area. The FormFX control gives you the ability to have multi-line captions as well as pictures.

1. Start with a new form and place the FormFX control anywhere on it.
2. Now in the **Caption** property of the FormFX control type "This is a FormFX Sample Showing MultiLine Captions".
3. Set **CaptionMultiLine** property to **True**. This will cause the FormFX control to wrap text in the caption if necessary. If you want to force a line breaks at runtime separate each line in the FormFX control's **Caption** with carriage return/line feeds (Chr$(13) + Chr$(10)). For example:

```
Sub Form_Load ()

    Dim crlf As String
    crlf = Chr$(13) + Chr$(10)
    Fx1.Caption = "Line 1" + crlf + "Line 2" + crlf + "Line 3"
End Sub
```

4. Set the **AlignmentCaption** property to '1 - Left Justify - MIDDLE' so that the caption starts at the left of the caption area.
5. Now let's display an icon at the left of the text.   First, select an icon in the **CaptionPicture** property (for this exercise, we will use the icon called BULLSEYE.ICO found in the \VB\ICONS\MISC directory).
6. To get the icon to appear left of the text, set the **AlignmentPicture** property to '9 - Left of Text'.
7. Now we just need to make the caption area large enough to show multi-line text and the icon. Set the **CaptionHeight** property to 1000.
8. Run the program. Adjust the width of the form and watch how the caption automatically 'wraps' to fit within the caption area.



Exercise 6: Changing Runtime Behavior

This exercise will show you how to use FormFX to change the behavior of a form such as minimizing and maximizing the size of the form, locking the form's size entirely and preventing the form for moving. Also, we will show you an example of a form with no caption that can still be moved with the mouse!

1. Start with a new form and place the FormFX control anywhere on it.
2. First we will set the FormFX control's properties that control the minimum size of a form. Set the both **SizeMinWidth** and **SizeMinHeight** properties to 2500. Now run the program and adjust the size of the form to its smallest size. Notice that the FormFX control has restricted you from sizing the form less than 2500 by 2500.

   Setting a maximum width and height is just as easy. Set the **SizeMaxWidth** and **SizeMaxHeight** properties to 3500. Then run the program and see how large you can size the form.
3. Now let's prevent the form from being resized at all by setting the **LockSize** property to **True**. When you run the program, you will see that you can no longer size the form at all. In addition to the **LockSize** property, the **LockMove** property disables movement of the form. Set the **LockMove** property to **True,** run the program and try to move the form.
4. Now even though we set the **LockSize** and **LockMove** properties to **True** in the previous step, you can hold down the Ctrl key to override the lock and resize or move the form. Give it a try. If you want, you can disallow the Ctrl key override by setting the **LockCtlKeyOverride** property to **False**.
5. Now, we will change the form so that it has no caption area but can still be moved. First, set the Form's **ControlBox, MinButton,** and **MaxButton** properties all to **False**. Then, set the **Caption** property on both the Form and FormFX control to blank. This will eliminate the caption area altogether.
6. Now if you run the program you will see that, by default, there is no way to move this form. Stop the program via the Visual Basic 'Run' menu. Now, set the FormFX's **ClientMove** property to **True**. Run the program again. You can now move the window by clicking anywhere on the client area of the form and dragging the mouse.

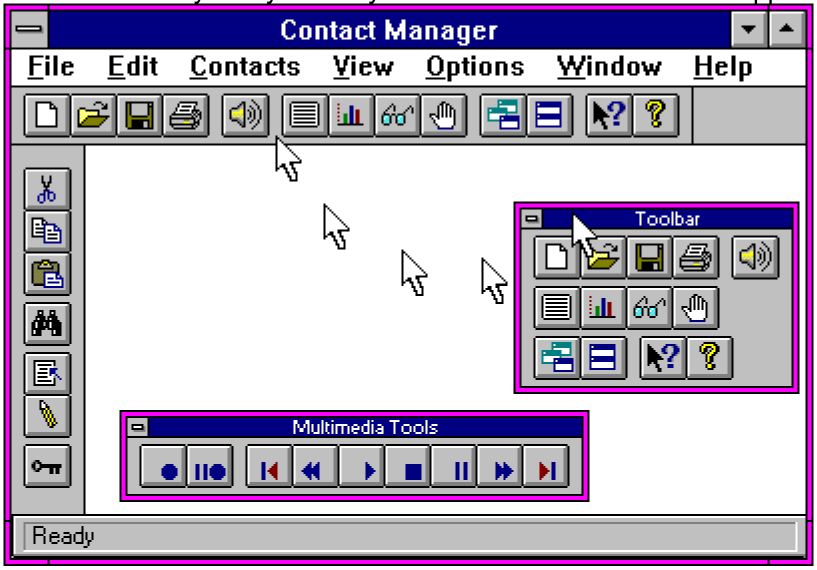
Well, that concludes our tour of the FormFX control. We showed you several different effects that the control can give you when customizing your forms. However, there is still much more. See Chapter 10 - 'The FormFX Control' for a complete list of properties and refer to Chapter 11 - 'Properties Reference' for a description of each one.

The Dockable Toolbar

Properties Events Methods

Description

The Dockable Toolbar is a custom control that lets a developer easily add toolbar functionality to an application while using a minimum of system resources.   In addition, on MDI forms, the toolbars can be allowed to float freely or to be docked to the top, bottom, left or right of the form. This is the same interface used by many of today's 'state of the art' commercial applications.



File Name

SSDOCKTB.VBX

ObjectType

SSToolbar

Related Topics:

- Properties Quick Reference
- Terms and Concepts
- Using the Toolbar Designer
- Creating a Toolbar at Runtime

Properties that apply only to this control are marked with an asterisk (*). For all other standard properties, see the *Microsoft Visual Basic Language Reference.*

* (About)
* (ToolbarDesigner)
Align
* AllowToolbarDockBottom
* AllowToolbarDockLeft
* AllowToolbarDockRight
* AllowToolbarDockTop
* AllowToolbarFloat
* AllowToolbarMove
BackColor
* BackColorDock
* BackColorHelp
* BalloonHelp
* BalloonHelpDelay
* BevelColorFace
* BevelColorHighlight
* BevelColorShadow
* BevelInner
* BevelOuter
* BevelWidth
* BorderWidth
* BtnEnabled
* BtnGroupSpacing
* BtnHeight
* BtnMarginX
* BtnMarginY
* Btns
* BtnSpacing
* BtnToolNum
* BtnValue
* BtnVisible
* BtnWidth
* DockRank
* DockRankSeq
* DockStatus
DragIcon
DragMode
Enabled
* FloatingCaption
* FloatingCaptionType
* FloatingControlBox
* FloatingHeight
* FloatingLeft
* FloatingSizable
* FloatingTop
* FloatingWidth
* FloatingWidthInBtns
FontBold
FontItalic
FontName
FontSize
FontStrikeThru
FontUnderline
* ForeColorHelp
Height
HelpContextID
Index
Left
MousePointer
Name
* Outline
Tag
* Toolbar3D
* ToolsetName
* ToolsetNumBtnStates
* ToolsetNumTools
* ToolsetPicture
* ToolsetToolAllowAllUp
* ToolsetToolDesc
* ToolsetToolExclusive
* ToolsetToolGroup
* ToolsetToolHelp
* ToolsetToolID
* ToolsetToolMnemonic
* ToolsetToolPicDnDis
* ToolsetToolPicDn
* ToolsetToolPicUp
* ToolsetToolPicUpDis
* ToolsetToolType
Top
Visible
Width

Events that apply only to this control are marked with an asterisk (*). For all other standard events, see the *Microsoft Visual Basic Language Reference*.

* Click

DblClick

* DockStatusChanged

DragDrop

DragOver

* Help

MouseDown

* MouseEnter

* MouseExit

MouseMove

MouseUp

* ToolbarClosed

* ToolbarResized

Drag Move Refresh

Terms and Concepts

Before continuing with the discussion of the Dockable Toolbar we need to explain the following terms and concepts:

- Toolbars and Buttons
- Toolset Objects
- Docked and Floating Toolbars

Related Topics:

Toolbars and Buttons
Toolset Objects
Docked and Floating Toolbars

Toolbars and Buttons

A toolbar is a single control that contains one or more tools. Each tool performs a specific function and is represented by a button.

There are two types of buttons:

- **Push buttons** - perform a specific function and then always return to their up state. An example of this type would be a button that invokes a print function.
- **Toggle buttons** - remain either in their up or down state.

These buttons can also be logically grouped together. A toolbar can contain multiple groups each consisting of one or more buttons. However, since the toolbar is actually a single control utilizing a single bitmap, the impact on system resources is greatly minimized.

In addition, toggle buttons in a group can operate independently or exclusively (with only one button in the group down at a time). For example, a group of three buttons might set an alignment option (left, center or right).   Only one of these buttons could be selected (in the down state) at a time.   If the 'left' button was down, pressing the 'center' button would cause the 'left' button to come up.

Every toolbar is linked to a toolset (via the ToolsetPicture property). Buttons for the toolbar are selected from the available tools in its toolset.

Both toolsets and toolbars are created and maintained using the Toolbar Designer (described later in this chapter).

Toolset Objects

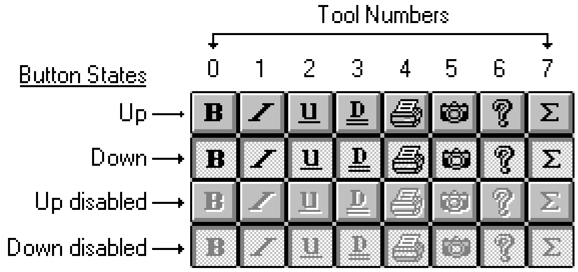
Toolsets are collections of related tools.   Each tool represents a specific user function.   Toolsets are defined and maintained in the Toolbar Designer. To define a toolset, you first supply a bitmap that contains an image for each state of every tool in the toolset. This bitmap is available at runtime via the ToolsetPicture property.

The possible states are:

1. Up (required)
2. Down (required)
3. Up disabled (optional. But required if 'Down disabled' is present.)
4. Down disabled (optional)

Note *The bitmap can be created or modified with any bitmap editor including the standard PaintBrush application (PBRUSH.EXE).*

The following bitmap contains images for all four states of 8 tools:



When creating the bitmap you need to be aware of the following:

- The size of the image for each state of every tool in the toolset must be the same.
- You must supply at least the 'up' and 'down states' (the 'up disabled' and 'down disabled' states are optional).
- The **ToolsetNumBtnStates** property lets you specify which button states are supplied. A value of two means that only the 'up' and 'down' states are included. Three means that the 'up disabled' state is also included while a value of four means that all four states are included (as in the sample bitmap above).
- The **ToolsetNumTools** property specifies how many tools are in the set. This would be set to 8 for the above sample bitmap.
- **BtnWidth** and **BtnHeight** properties are automatically calculated by the control based on the size of the bitmap and the **ToolsetNumBtnStates** and **ToolsetNumTools** property settings.

At design time, the Toolbar Designer lets you to specify **ToolsetName**, **ToolsetNumBtnStates**, **ToolsetNumTools** and several additional properties for each tool in the toolset.
At runtime, you can access these additional properties via the following property arrays (See Chapter 11 'Properties Reference' for more information about these properties):

- **ToolsetToolID**() - a unique ID (maximum 16 characters) that the developer can use internally to identify the tool.
- **ToolsetToolHelp**() - balloon help text.
- **ToolsetToolDesc**() - an expanded description of the tool which might be used in a status bar.
- **ToolsetToolType**()   - tool type, push ('0') or toggle ('1').

- **ToolsetToolGroup**() - affects both visual and functional aspects of the tools. Adjacent tools with the same group number are separated from each other based on the **BtnSpacing** property while adjacent tools with different group numbers are separated based on the **BtnGroupSpacing** property. For toggle type tools, the group number can also determine how they operate based on their **ToolsetToolExclusive**() and **ToolsetToolAllowAllUp**() property settings.
- **ToolsetToolExclusive**() - **True** or **False**. If set to **True** then all tools with the same **ToolsetToolGroup**() number as this tool will operate exclusively (with only one tool in the group down at a time). This applies only to tools with **ToolsetToolType**() set to toggle.
- **ToolsetToolAllowAllUp**() - **True** or **False**. If set to **True** allows all tools in an exclusive group to be toggled up. This applies only to tools with **ToolsetToolType**() set to toggle and **ToolsetToolExclusive**() set to **True**.
- **ToolsetToolMnemonic**() - an accelerator character (optional).

The indexes for these property arrays are zero based and correspond to each tool in the toolset. For example, in the sample bitmap above 0 represents the B tool, 1 represents the I tool and so on.

Note *All of these settings are actually stored in a bitmap information file that has the same name as the bitmap but with a BM$ extension. For example, if the bitmap was named TOOLSET1.BMP the above settings would be saved in the same directory in a file named TOOLSET1.BM$.*

The advantage to storing these settings in a parallel file with the bitmap is that you don't need to redefine these parameters for every form and/or project that uses the same toolset. In effect, they can be viewed as reusable objects.

Once you have created a toolset and assigned it to a toolbar, the Toolbar Designer provides an easy drag and drop method for arranging tools on the toolbar. Simply drag a tool from the toolset and drop it on the toolbar at the appropriate spot. An instance of a tool on the toolbar is referred to as a button.

For each button, the Toolbar Designer lets you specify whether at runtime the button is initially enabled, visible, up or down. These settings can also be controlled via the following runtime property arrays (See Chapter 11 - 'Properties Reference' for more information about these properties):

- **BtnToolNum**() - this identifies which tool this button represents. It is the zero based index of the tool in the toolset (described above).
- **BtnVisible**() - **True** or **False**
- **BtnEnabled**() - **True** or **False**
- **BtnValue**() - **True** or **False**, determines the state (up is **False**, down is **True**)

The indexes for these button property arrays are zero based and correspond to each button's relative position on the toolbar. The leftmost button is zero.

Docked and Floating Toolbars

When placed on a standard Visual Basic form (not an MDI form) or on a container control (e.g. a picturebox) the toolbar control is stationary. It has the standard Align property so that it can resize itself automatically when its container is resized.

However, when placed directly on an MDI form (not on a container) the toolbar can either float independently or be docked to the top, bottom, left or right docking area of the MDI form. In addition, multiple toolbars can share the same docking area.

Note *The four docking areas (top, bottom, left and right) are outside the MDI form's client area so they will not affect MDI Child forms.*

The following properties control docking and floating behavior and apply only to toolbars on MDI forms (See Chapter 11 'Properties Reference' for more information about these properties):

- **AllowToolbarDockTop** - **True** or **False**
- **AllowToolbarDockBottom** - **True** or **False**
- **AllowToolbarDockLeft** - **True** or **False**
- **AllowToolbarDockRight** - **True** or **False**
- **AllowToolbarFloat** - **True** or **False**
- **AllowToolbarMove** - **True** or **False**, if set to **False** the user can't move the toolbar at runtime.
- **DockStatus** - top, left, bottom, right or not docked (floating).
- **DockRank** and **DockRankSeq** - these properties determine the relative position of toolbar rows as well as the relative position of a toolbar with respect to other toolbars that are docked in the same docking area. For example, in the top docking area there may be 3 toolbars in 2 ranks.

Using the Toolbar Designer

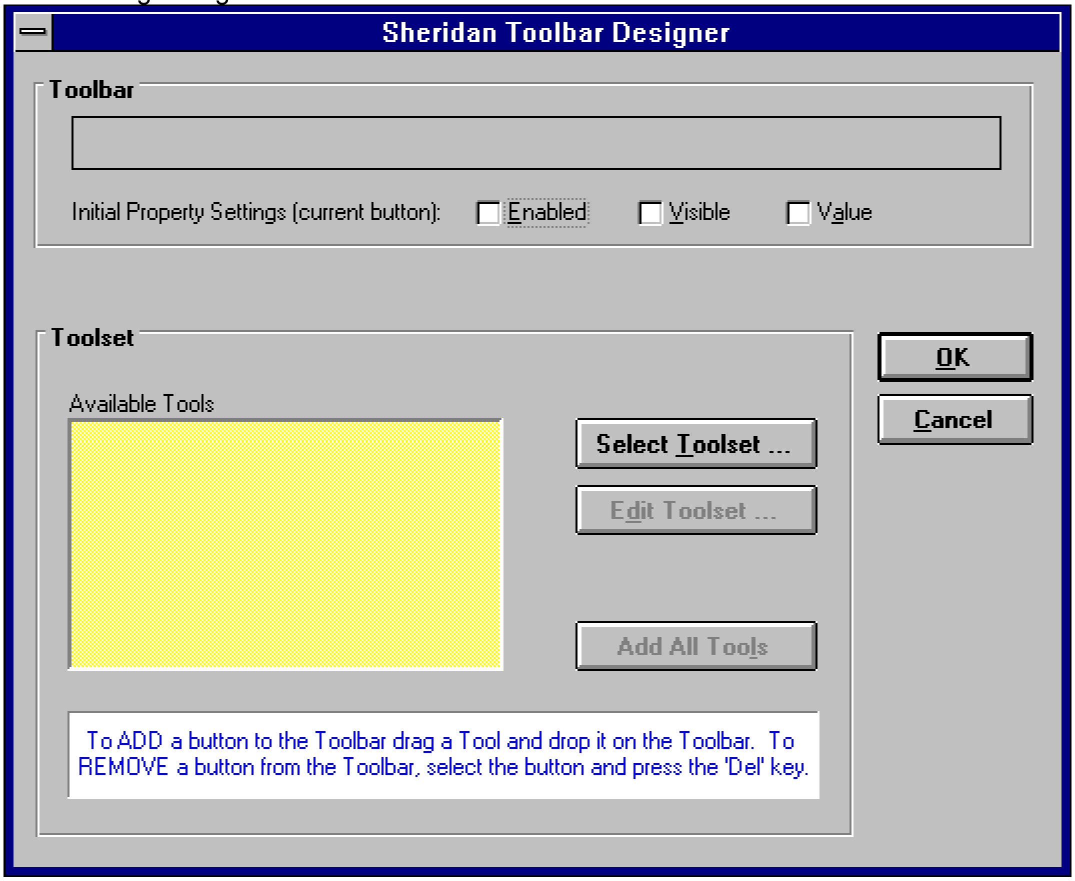
The Toolbar Designer is a design time tool for creating and maintaining toolsets and assigning tools to a toolbar. In this section we make the assumption that you understand these and other concepts.

Therefore, if you haven't read the previous section in this chapter ('Terms and Concepts') we recommend that you do so before continuing.

First, if you have not already done so, add the SSDOCKTB.VBX file to your project (see chapter 3 - Using Designer Widgets).

Next place a toolbar control on a form using the tool in the Visual Basic toolbox.

In the Visual Basic property window double click on the **(Toolbar Designer)** property. This will bring up the following dialog:

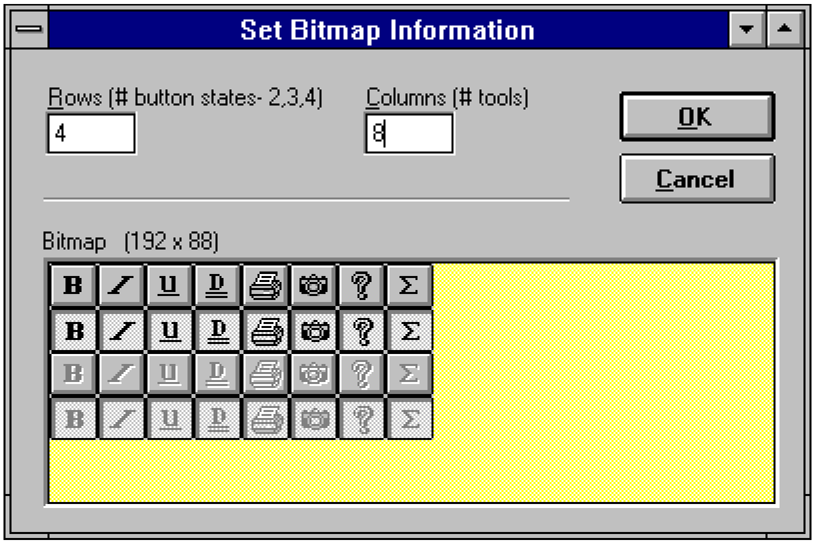


To ADD a button to the Toolbar drag a Tool and drop it on the Toolbar. To REMOVE a button from the Toolbar, select the button and press the 'Del' key.

Press the button labeled 'Select Toolset...' (if there was already a toolset attached to the toolbar this button would be labeled 'Change Toolset...'). This will bring up a file open dialog requesting the name of the bitmap file. If you haven't created your own bitmap you can use TOOLSET1.BMP in the SAMPLES\CHAPTER8 directory provided with Designer Widgets.

After selecting a BMP file, if there is no parallel BM$ file (see explanation of the bitmap information file in the previous section under the heading 'Toolset Objects') the 'Set Bitmap Information' dialog will be displayed:

Setting Bitmap Information



Rows (# button states- 2,3,4): 4

Columns (# tools): 8

Bitmap (192 x 88)

This dialog displays the bitmap along with its width and height in pixels.
From here you must supply the following information:

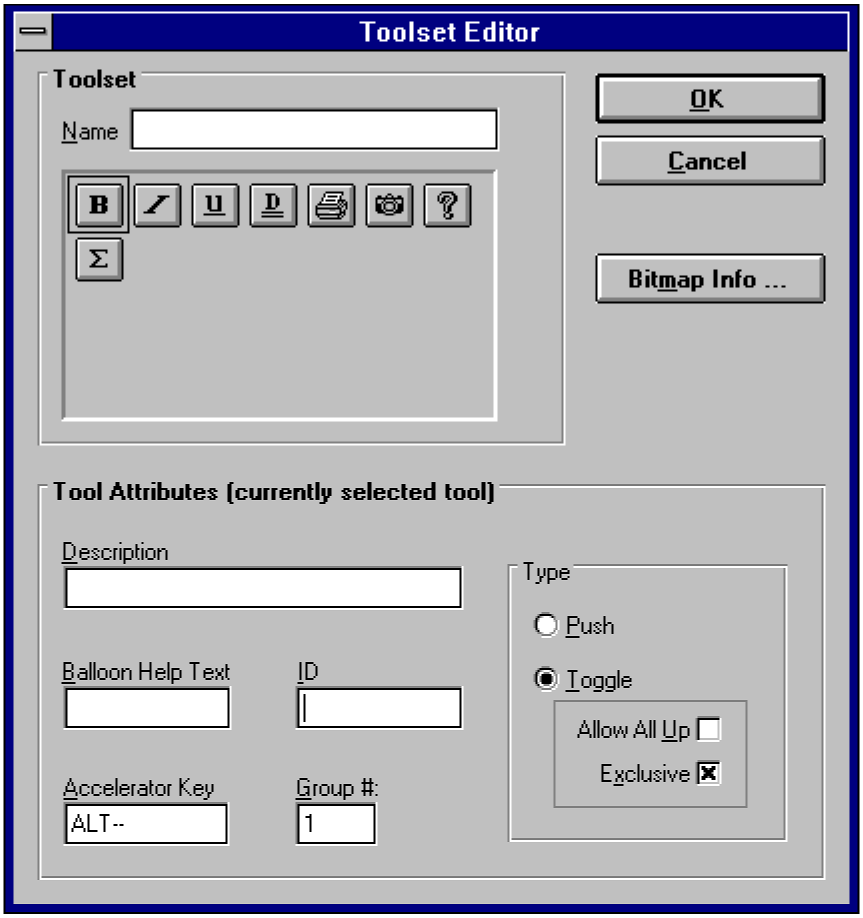
- Rows - corresponds to the **ToolsetNumBtnStates** property.
- Columns - corresponds to the **ToolsetNumTools** property.

It is important that these numbers correspond exactly to the number of rows and columns in the bitmap. Otherwise, you will end up with some very strange looking tools (See Chapter 11 - 'Properties Reference' for more information about these properties).

Pressing the 'OK' button will bring you back to the Toolbar Designer main window.   Now press the 'Edit Toolset' button.

The 'Toolset Editor' should look like this:

Setting Toolset Information



Name

Description

Balloon Help Text

ID

Accelerator Key: ALT--

Group #: 1

○ Push

● Toggle

Allow All Up □

Exclusive ☒

From here you specify the name of the toolset (this corresponds to the **ToolsetName** property).
You can also set several properties for each tool in the toolset.
Select a tool by clicking on it in the toolset box. The selected tool will be shown with a rectangular outline around it.
For the selected tool you can specify the following (see Chapter 11 - 'Properties Reference' for more information about these properties):

- **Description** - corresponds to the **ToolsetToolDesc()** property. This can be used to display an expanded description of the button in a status bar when the cursor is over the tool (see the **Help**, **MouseEnter** and **MouseExit** events in chapter 12).
- **Balloon Help Text** - corresponds to the **ToolsetToolHelp()** property. This text is displayed in a balloon help window when the cursor pauses over the tool at runtime (see the **BalloonHelp** and **BalloonHelpDelay** properties in chapter 11 and the **Help** event in chapter 12).
- **ID** - corresponds to the **ToolsetToolID()** property. This ID can be a maximum of 16 characters. It uniquely identifies the tool and is passed as the first parameter in the **Click**, **Help**, **MouseEnter** and **MouseExit** events.
- **Accelerator Key** - corresponds to the **ToolsetToolMnemonic()** property. If the form (that the toolbar is on) is the active window, holding the ALT key down and pressing this character will push or toggle this tool. Remember, that if an MDI Child form is the active form, accelerator keys for toolbars on the main MDI form will NOT be active.
- **Group #** - corresponds to the **ToolsetToolGroup()** property. This property allows you to logically group tools. Adjacent tools on a toolbar with the same group # will be separated based on the **BtnSpacing** property while adjacent tools with different group #'s will be separated based on the **BtnGroupSpacing** property. Also toggle

type tools within the same group can behave differently based on the 'Allow All Up' and 'Exclusive' settings.

- **Type** - corresponds to the **ToolsetToolType()** property (either push or toggle).
- **Exclusive** - corresponds to the **ToolsetToolExclusive()** property. If checked will cause all of the toggle type tools with the same group # as this tool to operate as a mutually exclusive group. In other words only one tool in the group can be toggled down at any time.
- **Allow All Up** - corresponds to the **ToolsetToolAllowAllUp()** property. If checked will allow all toggle type tools with the same group # as this tool to be toggled up at the same time. This setting only has meaning if **Exclusive** is also checked.

Note *Changing the 'Exclusive' or 'Allow All Up' settings for one tool in a group will automatically cause these settings on all the other tools in the same group to be changed as well.*

The 'Bitmap Info...' button will display the 'Set Bitmap Information' dialog (explained previously in this section).

Press 'OK' to save any changes or press 'Cancel' to exit without saving them.

The Toolbar Designer main window should now be shown with the tools from the selected toolset displayed in a box labeled 'Available Tools'.

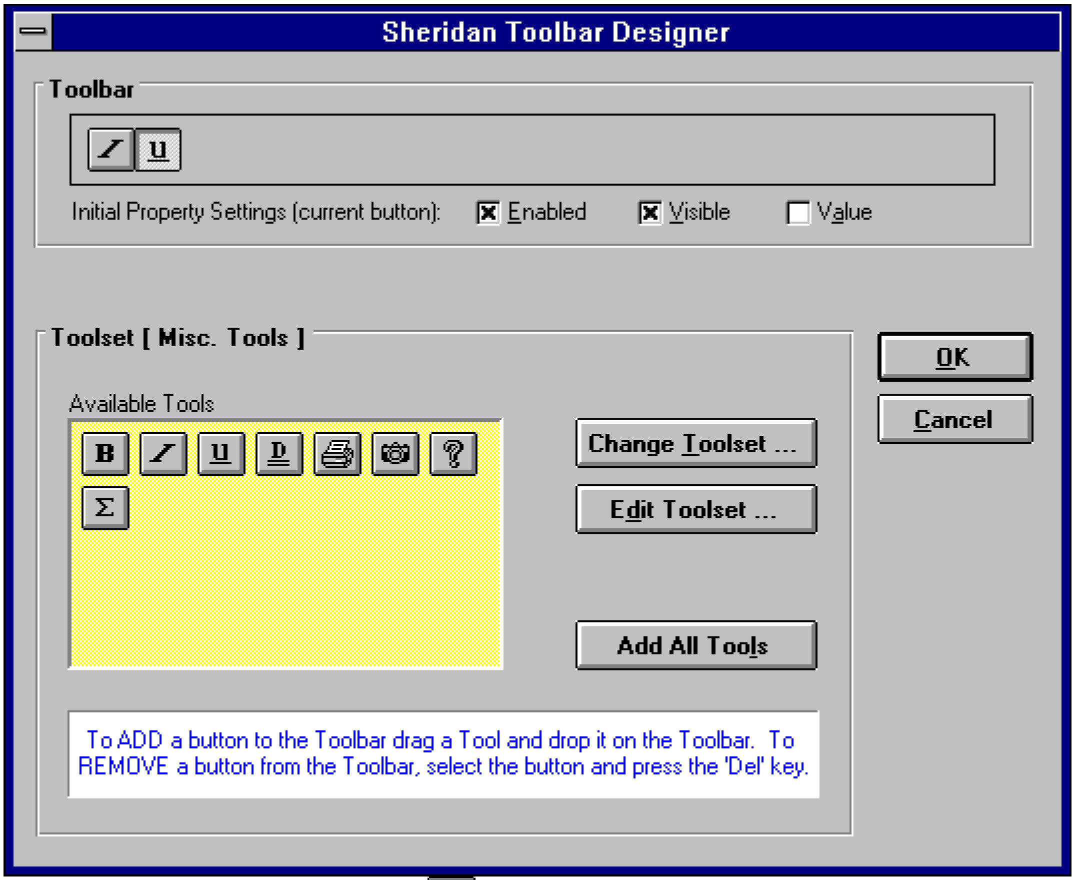
To add a tool to the toolbar simply drag it from the 'Available Tools' box and drop it on the toolbar.

To insert a tool between 2 other tools drop it between them on the toolbar.

To remove a tool click on it in the toolbar (not in the 'Available Tools' box) and press the DELETE key. A dialog will appear asking you to confirm the deletion.

Press 'Add All Tools" to add all of the tools in the toolset to the toolbar. This will overlay any tools that were already on the toolbar.

In addition, there are 3 check boxes labeled 'Initial Property Settings'. They apply to the currently selected button. You select the currently selected button by clicking on it in the toolbar.



Initial Property Settings (current button): ☒ Enabled ☒ Visible □ Value

To ADD a button to the Toolbar drag a Tool and drop it on the Toolbar. To REMOVE a button from the Toolbar, select the button and press the 'Del' key.

In the above example the underline U tool is selected and the 'Initial Property Settings' are set so that when the toolbar is first displayed the underline tool will be enabled, visible and will be in its up state (not selected).

These check boxes correspond the **BtnEnabled()**, **BtnVisible()** and **BtnValue()** properties respectively.

Note *Although the **BtnToolNum()** property is not explicitly shown here it is automatically set based on the index of the tool that was dragged from the 'Available Tools' box.*

Press 'OK' to save any changes or press 'Cancel' to exit without saving them.

Creating a Toolbar at Runtime

Creating toolsets and toolbars at design time using the Toolbar Designer requires no coding on your part.

However, there are situations where it might be necessary to dynamically create a toolbar and its underlying toolset. For example, suppose you wanted to give the user the flexibility of building a custom toolbar at runtime. You might want to offer the user a choice of tools from a number of toolsets. This is possible since all of the properties that relate to toolsets and toolbars are modifiable from code.

In this section will we describe how to dynamically create a toolbar and its toolset at runtime.

There are 2 ways of creating the toolset picture at runtime:

- set the individual pictures for each state of every tool via the **ToolsetToolPicUp()**, **ToolsetToolPicDn()**, **ToolsetToolPicUpDis()** and **ToolsetToolPicDnDis()** properties.
- set ToolsetPicture**()** to a bitmap containing all the states for every tool.

Note that either method requires that **ToolsetNumTools** and **ToolsetNumBtnStates** also be set.

The following code template shows how create a tooolset from individual pictures. It would normally be placed in the **Load** event of the form but could also be executed at any time. In this example the **Name** of the SSToolbar control is 'Tb1'. The variables that you need to supply are enclosed in brackets [].

```
Dim I As Integer
'first clear any previous toolset picture
Tb1.ToolsetPicture = LoadPicture("")
'set the name for toolset (optional)
Tb1.ToolsetName = [descriptive name]
'then create the toolset using individual pictures
Tb1.ToolsetNumBtnStates = [# of states(2, 3 or 4)]
Tb1.ToolsetNumTools = [# of tools in toolset]
For i = 0 to Tb1.ToolsetNumTools - 1
        Tb1.ToolsetToolID(i) = [unique 16 char ID]
        Tb1.ToolsetToolType(i) = [0 = push, 1 = toggle]
        Tb1.ToolsetToolDesc(i) = [status bar desc]
        Tb1.ToolsetToolHelp(i) = [balloon help text]
        Tb1.ToolsetToolMnemonic(i)= [accelerator char]
        Tb1.ToolsetToolGroup(i) = [group #]
        Tb1.ToolsetToolExclusive(i) = [True/False]
        Tb1.ToolsetToolAllowAllUp(i) = [True/False]
                ' Individual tool pictures
                ' see note below about
                ' picture properties
        Tb1.ToolsetToolPicUp(i) = [up bitmap]
        Tb1.ToolsetToolPicDn(i) = [down bitmap]
                'if ToolsetNumBtnStates>2
        Tb1.ToolsetToolPicUpDis(i) = [up disabled bitmap]
                'if ToolsetNumBtnStates>3
        Tb1.ToolsetToolPicDnDis(i) = [down disabled bitmap]
Next i
'attach the tools to buttons
Tb1.Btns = [# of buttons in toolbar]  'does not have to be
        'the same as the #
        'of tools in the toolset
For i = 0 to Tb1.Btns - 1
        Tb1.BtnToolNum(i) = [index of tool in toolset array]
        Tb1.BtnVisible(i) = [True/False]
        Tb1.BtnEnabled(i) = [True/False]
        Tb1.BtnValue(i) = [False is up, True is down]
Next I
```

Note *After setting **ToolsetNumBtnStates** and **ToolsetNumTools** properties, the first individual tool picture you set will cause **BtnWidth** and **BtnHeight** to be automatically calculated by the control based on the picture's size (to subsequently change **BtnWidth** and **BtnHeight**, you must first reset the ToolsetPicture property via LoadPicture("")).*

*Remember that instead of setting the individual pictures for each state of every button, as in the above example, you can alternatively set the **ToolsetPicture** property to a single bitmap that contains all of these images.*

*Actually, the individual picture properties **ToolsetToolPicUp()**, **ToolsetToolPicDn()**, **ToolsetToolPicUpDis()** and **ToolsetToolPicDnDis()** are not kept as separate bitmaps by the toolbar. They are used just like the **GraphicCell()** property in the Picture Clip control (included in the Visual Basic Professional Edition) except that they are not read-only.*

*Keep in mind that when you set any of these individual picture properties you are in fact updating an area of the larger (all inclusive) **ToolsetPicture** bitmap. By taking this segmented approach we minimize the impact on system resources.*

Here is another example of how you might use these properties. Suppose you had a toolbar ('Tb1') on one form ('F1') and you wanted to let the user add a tool to it from another toolbar ('Tb2') on another form ('F2') and the tool that you wanted to add was the fourth button from the left on Tb2.

Here is how you would do it:

```
Dim ExistingTool as Integer
Dim NewTool as Integer
Dim NewButton as Integer
ExistingTool = F2!Tb2.BtnToolNum(3)   'get tool index of
                                      '4th button in Tb2
'add a button to the toolbar on Form 1
NewTool = F1!Tb1.ToolsetNumTools      'total tools in Tb1
F1!Tb1.ToolsetNumTools = NewTool + 1  'add a tool to Tb1
                                      'set up and down tool images
F1!Tb1.ToolsetToolPicUp(NewTool) = F2!Tb2.ToolsetToolPicUp(ExistingTool)
F1!Tb1.ToolsetToolPicDn(NewTool) = F2!Tb2.ToolsetToolPicDn(ExistingTool)
'if ToolsetNumBtnStates > 2 set up disabled tool image
If (F1!Tb1.ToolsetNumTools > 2) and (F2!Tb2.ToolsetNumTools > 2) then
        F1!Tb1.ToolsetToolPicUpDis(NewTool) = F2!Tb2.ToolsetToolPicUpDis(ExistingTool)
End If
'if ToolsetNumBtnStates > 3 set down disabled tool image
If (F1!Tb1.ToolsetNumTools > 3) and (F2!Tb2.ToolsetNumTools > 3) then
        F1!Tb1.ToolsetToolPicDnDis(NewTool) = F2!Tb2.ToolsetToolPicDnDis(ExistingTool)
End If
'set remaining properties
F1!Tb1.ToolsetToolID(NewTool) = F2!Tb2.ToolsetToolID(ExistingTool)
F1!Tb1.ToolsetToolType(NewTool) = F2!Tb2.ToolsetToolType(ExistingTool)
F1!Tb1.ToolsetToolDesc(NewTool) = F2!Tb2.ToolsetToolDesc(ExistingTool)
F1!Tb1.ToolsetToolHelp(NewTool) = F2!Tb2.ToolsetToolHelp(ExistingTool)
F1!Tb1.ToolsetToolMnemonic(NewTool) = F2!Tb2.ToolsetToolMnemonic(ExistingTool)
F1!Tb1.ToolsetToolGroup(NewTool) = F2!Tb2.ToolsetToolGroup(ExistingTool)
F1!Tb1.ToolsetToolExclusive(NewTool) = F2!Tb2.ToolsetToolExclusive(ExistingTool)
F1!Tb1.ToolsetToolAllowAllUp(NewTool) = F2!Tb2.ToolsetToolAllowAllUp(ExistingTool)
'add a button to the toolbar on Form 1
NewButton = F1!Tb1.Btns       'total Btns in Tb1
F1!Tb1.Btns = NewButton + 1   'add a Btn to Tb1
F1!Tb1.BtnToolNum(NewButton) = NewTool
F1!Tb1.BtnVisible(NewButton) = True
F1!Tb1.BtnEnabled(NewButton) = True
F1!Tb1.BtnValue = False
```

Note *The above example assumes that the dimensions of each of the tools (**BtnWidth** and **BtnHeight**) in both toolsets are the same. If they aren't, the new tool might not look right. Therefore, if you plan on providing this type of facility in your applications (moving or copying tools between toolbars) you will need to standardize on a single size for your buttons.*

If you would like to see some more sample code look at the projects that are prefixed with TBAR in the SAMPLES directory.

Properties Quick Reference

This section will give you a quick reference to all properties grouped into functional categories.   For a complete reference of all properties, see Chapter 11.   If the letter 'r' appears next to a property, the property is only available at runtime.

Related Topics:

Docking Properties (MDI forms only)
Floating Properties (MDI forms only)
Button Properties
Toolbar Properties
Toolset Properties
3D Properties
Font Properties
Sizing/Location Properties
Other Properties

Docking Properties **(MDI forms only)**

AllowToolbarDockBottom	Allows the toolbar to be docked in the bottom docking area.
AllowToolbarDockLeft	Allows the toolbar to be docked in the left docking area.
AllowToolbarDockRight	Allows the toolbar to be docked in the right docking area.
AllowToolbarDockTop	Allows the toolbar to be docked in the top docking area.
AllowToolbarFloat	Allows the toolbar to float.
AllowToolbarMove	Allows the toolbar to be moved by the user at runtime.
BackColorDock	Specifies the color of the docking areas.
DockRank	Determines which rank (i.e. 'row' for top and bottom docking areas or 'column' for left and right docking areas) of a docking area that the toolbar is docked in.
DockRankSeq	Determines the relative position of a toolbar with respect to other toolbars that are docked in the same rank in a docking area.
DockStatus	Determines if the toolbar is docked to the top, bottom, right or left docking area or is not docked at all (floating).

Floating Properties **(MDI forms only)**

FloatingCaption	Sets the caption for the floating toolbar.
FloatingCaptionType	Specifies the type of caption (none, small, normal or use font).
FloatingControlBox	Determines if a control box will appear in the caption area of the floating toolbar.
FloatingWidth	Determines the width of the floating toolbar in units based on the scalemode of the MDI form and relative to the left of the MDI client area.
FloatingLeft	Determines the X-coordinate of the floating toolbar in units based on the scalemode of the MDI form and relative to the left of the MDI client area.
FloatingSizable	Determines whether the user can resize a floating toolbar.
FloatingTop	Determines the Y-coordinate of the floating toolbar in units based on the scalemode of the MDI form and relative to the top of the MDI client area.
FloatingHeight	Determines the height of the floating toolbar in units based on the scalemode of the MDI form and relative to the left of the MDI client area.
FloatingWidthInBtns	Specifies the width of the floating toolbar in number of buttons.

Button Properties

BtnEnabled() - [r] Determines if this button is enabled.

BtnHeight - [r] The height (in pixels) of all buttons on the toolbar. This property is automatically calculated based on the size of the bitmap and **ToolsetNumBtnStates.**

Btns - [r] Determines the # of buttons on the toolbar.

BtnToolNum() - [r] Determines which tool in the toolset this button represents (valid values are from -1 to **ToolsetNumTools** - 1). It is passed as a parameter in the **Click**, **Help**, **MouseEnter** and **MouseExit** events.

BtnValue() - [r] Determines which state the button is in (**True** = down, **False** = up).

BtnVisible() - [r] Determines if this button is visible.

BtnWidth - [r] The width (in pixels) of all buttons on the toolbar. This property is automatically calculated based on the size of the bitmap and **ToolsetNumTools.**

Toolbar Properties

Align Standard Visual Basic align property.

BackColor Determines the color of the toolbar. It has meaning only if Toolbar3D is set to **False**. Otherwise, **BevelColorFace** is used.

BackColorHelp Determines the background color used in balloon help.

BalloonHelp Determines if balloon help is on. If set to **True** then pausing over a button at runtime will cause the **Help** event to be fired and its **ToolsetToolHelp()** text to be displayed.

BalloonHelpDelay Determines the amount of time (in milliseconds) the user can pause over a button before the **Help** event is fired and balloon help is displayed.

BtnGroupSpacing Determines the number of pixels that separate adjacent buttons that have different **ToolsetToolGroup()** numbers.

BtnMarginX Determines the number of pixels that separate the buttons from the left edge of the toolbar or from the inside bevels (if **Toolbar3D** is **True**).

BtnMarginY Determines the number of pixels that separate the buttons from the top edge of the toolbar or from the inside bevels, if (**Toolbar3D** is **True**).

BtnSpacing Determines the number of pixels that separate adjacent buttons that have the same **ToolsetToolGroup()** number.

ForeColorHelp Determines the forecolor (color of the text) used in balloon help.

Outline Draws an outline (single border) around the toolbar when docked or stationary.

Toolset Properties

ToolsetName - [r] A descriptive name for the toolset. This might be used for presenting the user with a choice of possible toolsets.

ToolsetNumBtnStates - [r] Determines how many button states (rows) are in the bitmap. This can range from 2 to 4.

ToolsetNumTools - [r] Determines how many tools (columns) are in the bitmap.

ToolsetPicture - [r] Gets or sets the toolset bitmap.

ToolsetToolAllowAllUp() - [r] Allows all toggle type tools, with the same **ToolsetToolGroup** # as this tool, to be toggled up at the same time.

ToolsetToolDesc() - [r] Specifies an expanded description of the tool.

ToolsetToolExclusive() - [r] Determines if all toggle type tools, with the same **ToolsetToolGroup** # as this tool, will operate exclusively (only one down at a time).

ToolsetToolGroup() - [r] Determines which group the tool belongs to.

ToolsetToolHelp() - [r] Specifies a short description of the tool (used by balloon help).

ToolsetToolID() - [r] Specifies an identifier (maximum of 16 characters) which can be referenced in VB code to uniquely identify the tool.

ToolsetToolMnemonic() - [r] Specifies an accelerator character that can be pressed by the user to select this tool.

ToolsetToolPicDnDis() - [r] Sets or gets a bitmap containing the down disabled state of the tool (applies only if **ToolsetNumBtnStates** > 3).

ToolsetToolPicDn() - [r] Sets or gets a bitmap containing the down state of the tool.

ToolsetToolPicUp() - [r] Sets or gets a bitmap containing the up state of the tool.

ToolsetToolPicUpDis() - [r] Sets or gets a bitmap containing the up disabled state of the tool (applies only if **ToolsetNumBtnStates** > 2).

ToolsetToolType() - [r] Specifies the type of the tool (0 = push, 1 = toggle).

3D Properties

BevelColorFace Sets the color for the toolbar surface. Applies only if **Toolbar3D** is **True.**

BevelColorHighlight Sets the highlight color of the toolbar bevel. Applies only if **Toolbar3D** is **True.**

BevelColorShadow Sets the shadow color of the toolbar bevel. Applies only if **Toolbar3D** is **True.**

BevelInner Specifies the type of inside bevel for the toolbar. Applies only if **Toolbar3D** is **True.**

BevelOuter Specifies the type of outside bevel for the toolbar. Applies only if **Toolbar3D** is **True.**

BevelWidth Sets the width of the bevel of the toolbar. Applies only if **Toolbar3D** is **True.**

BorderWidth Sets the amount of space in pixels between the inner and outer bevels. Applies only if **Toolbar3D** is **True.**

Toolbar3D If set to **True** ignores **BackColor** and uses to above properties to give a 3D look to the toolbar.

Font Properties

FontBold	Display text bold.
FontItalic	Display text italic.
FontName	Specifies the font name.
FontSize	The size of the font.
FontStrikeThru	Display text with strike through.
FontUnderline	Display text with underlines.

Sizing/Location Properties

Height	The height of the control.
Left	X coordinate of the left of control.
Top	Y coordinate of the top of control.
Width	The width of the control.

Other Properties

(About)	Displays version information about the control at design time.
(ToolbarDesigner)	Invokes the Toolbar Designer at design time.
DragIcon	Standard Visual Basic DragIcon property.
DragMode	Standard Visual Basic DragMode property.
Enabled	Determines if the control is enabled.
HelpContextID	Standard Visual Basic HelpContextID property.
Index	The index of the control if it is in a control array.
MousePointer	Standard Visual Basic MousePointer property.
Name	The name of the control.
Tag	The tag for the control.
Visible	Determines if the control is visible.

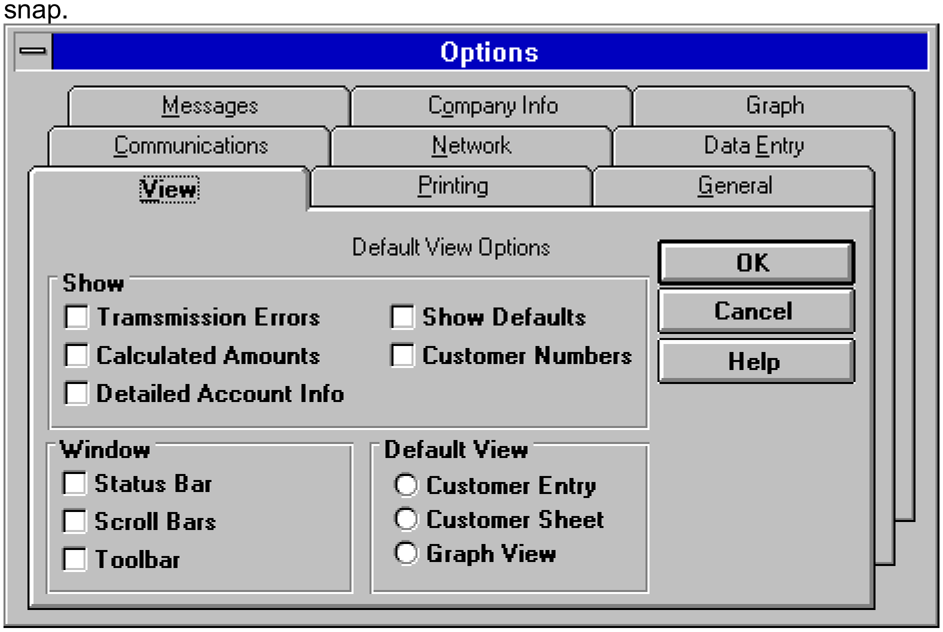
The Index Tab Control

Properties Events Methods

Description

The Index Tab is a custom control that you can use to provide the same 'state of the art' interface adopted by many of today's commercial Windows applications. This interface is based on an index card metaphor. This allows the presentation of several dialogs or screens of information from a single form. In addition, since this is a metaphor that everyone is familiar with, users feel comfortable with it immediately.

Tabs are selected (in design and run modes) by clicking on the tab with the left mouse button. When a tab is selected at runtime, the controls associated with it appear and the user can then view or modify the information. Since the tabs are also active at design time, creating and arranging each tab's controls is a snap.



Default View Options

Show
□ Tramsmission Errors
□ Show Defaults
□ Calculated Amounts
□ Customer Numbers
□ Detailed Account Info

Window
□ Status Bar
□ Scroll Bars
□ Toolbar

Default View
○ Customer Entry
○ Customer Sheet
○ Graph View

File Name

SSIDXTAB.VBX

ObjectType

SSIndexTab

Related Topics:

- Properties Quick Reference
- Placing Controls on the Index Tab
- Setting Index Tab Captions and Pictures at Design Time
- How to Setup the Number of Tabs
- Sharing Controls Across Different Tabs
- Control Limitations

Properties that apply only to this control are marked with an asterisk (*). For all other standard properties, see the *Microsoft Visual Basic Language Reference*.

* (About)
* ActiveTabBackColor
* ActiveTabFontBold
* ActiveTabFontItalic
* ActiveTabFontStrikeThru
* ActiveTabFontUnderline
* ActiveTabForeColor
* ActiveTabPicture

Align

* AlignmentCaption
* AlignmentPicture

BackColor

* BevelColorFace
* BevelColorHighlight
* BevelColorShadow
* Caption
* ClientLeft
* ClientHeight
* ClientTop
* ClientWidth

DragIcon

DragMode

Enabled

* Font3D

FontBold

FontItalic

FontName

FontSize

FontStrikeThru

FontUnderline

ForeColor

Height

HelpContextID

Index

Left

MousePointer

Name

* Picture
* Redraw
* Rows
* ShowFocusRect
* Tab
* TabBackColor
* TabCaption
* TabCutSize
* TabEnabled
* TabForeColor
* TabHeight
* TabHwnd

TabIndex

* TabMaxWidth
* TabOrientation
* TabPicture
* TabPictureMetaHeight
* TabPictureMetaWidth
* TabRowOffset
* Tabs
* Tabs3D
* TabSelectType
* TabsPerRow
* TabStart

TabStop

* TabVisible

Tag

Top

Visible

Width

Events that apply only to this control are marked with an asterisk (*). For all other standard events, see the *Microsoft Visual Basic Language Reference.*

- * Click
- DblClick
- DragDrop
- DragOver
- GotFocus
- KeyDown
- KeyPress
- KeyUp
- LostFocus
- MouseDown
- MouseMove
- MouseUp

Drag
Move
Refresh
SetFocus

Placing Controls on the Index Tab

The Index Tab control allows you to visually setup the controls that will appear on each tab within the Visual Basic design time environment. This is done simply by selecting each tab (using the left mouse button) one at a time and placing the desired controls directly onto the Index Tab control.

Note *Be sure that all controls placed on the tab are, in fact, child controls of the Index Tab. This is how the Index Tab control knows which controls are on which tab. To ensure that a control is a child of the Index Tab, first select the Index Tab control by clicking on it once with the left mouse button. Then select the desired tool from the Visual Basic toolbox. Now drag the control to the proper size directly on the Index Tab control.*

To test whether the controls are children of the Index Tab control, move the Index Tab control to a different location on the form. The controls should follow the Index Tab control. If they do not, refer to the Microsoft Visual Basic Programmer's Guide*, Chapter 3 - '*Grouping Options with Option Buttons'; 'Containers for Controls' *on page 46 for more information on parent-child relationships with controls*.

When finished with one tab, select the next tab. You will notice that all the controls that you placed on the previous tab have disappeared. The Index Tab control remembers where these controls were placed so that the next time you select the tab, the controls will reappear.

Setting Index Tab Captions and Pictures at Design Time

Two properties are used to set the captions and pictures of each index tab, they are the **TabPicture()** and **TabCaption()** properties. However, since these properties are *property arrays,* they are not available at design time. Therefore, we have provided a way to set these properties at design time with the **Caption** and **Picture** properties.

To set the caption and/or picture for a specific tab, first select the tab you want to set.   Then set the **Caption** property. The Index Tab control will automatically set the selected tab's **TabCaption()** property for you.   To set a picture for the selected tab, set the **Picture** property and the control will automatically set the **TabPicture()** property for the selected tab. Repeat this process for all the tabs.

When you save the form, all of these property settings will be written to disk.   The next time you load the project, the captions and pictures will be retrieved and set to the appropriate tabs.

How to Setup the Number of Tabs

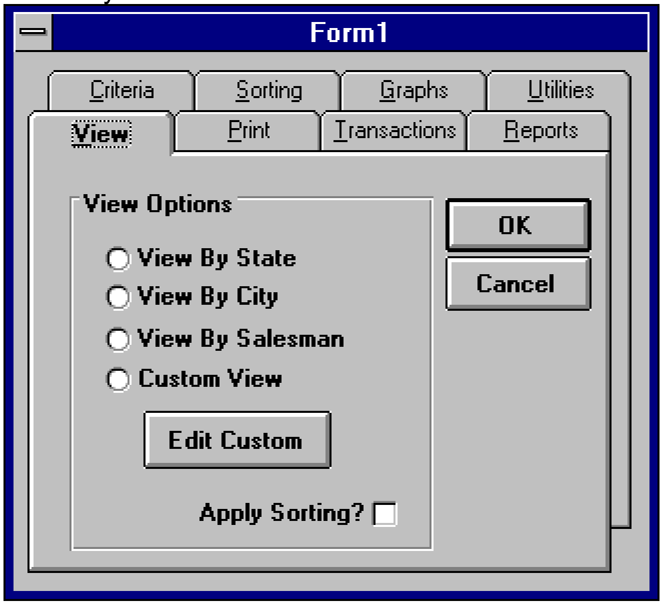
The Index Tab control allows you to specify the number of tabs to display on the control as well as define the number of rows of tabs. This involves the settings of two properties called **Tabs** and **TabsPerRow**. To set the overall number of tabs on the control, use the **Tabs** property. To determine how many rows of tabs to display, you need to specify a value for the **TabsPerRow** property. The Index Tab control will divide the number of tabs by the number of tabs per row to get the number of rows (**Tabs / TabsPerRow**). By default, the **TabsPerRow** property is set to 3. So if you set the **Tabs** property to 9, you will see 3 rows of tabs (9 / 3 = 3).

Furthermore, when the control calculates the number of rows of tabs, it applies the remainder of tabs to a new row. Therefore if the **Tabs** property is set to 8 and you set the **TabsPerRow** property to 3, the number of rows would still be 3. The first two rows would contain 3 tabs but the last row would only contain 2.

Sharing Controls Across Different Tabs

Using an Index Tab control with many tabs and many controls on each tab may sometimes use up scarce resources in Windows.   However, with the Index Tab control you can share common controls across several different tabs very easily.

The **TabHwnd()** property is provided for this purpose. This property array consists of a window handle for each of the tabs you define in the Index Tab control. To use the same control on several different tabs, you simply need to set the control's parent to the **TabHwnd()** property associated with the new tab.   The sample code below shows how to use the same OK and Cancel buttons (see figure below) on all the tabs for an Index Tab control.   It responds to the **Click** event by changing the parent window of the buttons to the newly selected tab.



```
Declare Function SetParent Lib "USER.EXE" (ByVal hWndChild As Integer, ByVal hWndParent As
Integer) As Integer

Sub SSIndexTab_Click (PreviousTab As Integer)
   Dim NewTab As Integer
   Dim rc As Integer
   NewTab = SSIndexTab.Tab
   rc = SetParent(cmdOK.Hwnd, SSIndexTab.TabHwnd(NewTab))
   rc = SetParent(cmdCancel.Hwnd, SSIndexTab.TabHwnd(NewTab))
End Sub
```

Control Limitations

Due to the nature of graphical controls they cannot be placed directly onto the Index Tab control. Graphical controls are controls in Visual Basic that do not have a window handle associated with them. You can normally determine if a control is graphical if it does not have an **Hwnd** property.   The Label, Line, Shape and Image controls are graphical.

If you need to use a graphical control on an index tab, you must place it inside a container control on the tab. A container control is a control in Visual Basic that can have child controls. The Picture, Frame and 3D Panel controls are containers. If you do not place graphical controls inside containers before placing them on a tab, the graphical controls will not disappear when a new tab is selected.

Properties Quick Reference

This section will give you a quick reference to all properties grouped into functional categories. For a complete reference of all properties, see Chapter 11. If the letter 'r' appears next to a property, the property is only available at runtime.

Related Topics:

Caption Properties
3D Effects Properties
Color Properties
Font Properties
Tab Properties
Sizing/Location Properties
Picture Properties
Other Properties

Caption Properties

AlignmentCaption	Determines how the caption will be aligned on the index tab.
Caption	Sets the text for the selected index tab at design time.
TabCaption() - [r]	Sets the caption for each tab.

3D Effects Properties

BevelColorFace	Sets the color of the face of the bevel.
BevelColorHighlight	Sets the color of the bevel highlight.
BevelColorShadow	Sets the color of the bevel shadow.
Tabs3D	Displays the index tabs in 3D.
Font3D	Displays the text in 3-D.

Color Properties

ActiveTabBackColor	Sets the background color of the active index tab.
ActiveTabForeColor	Sets the foreground color of the active index tab.
BackColor	Sets the background color of area not occupied by the control.
TabBackColor() - [r]	Sets the background color of each tab.
TabForeColor() - [r]	Sets the foreground color of each tab.

Font Properties

ActiveTabFontBold Display bold text on the active tab.

ActiveTabFontItalic Display italic text on the active tab.

ActiveTabFontStrikeThru Displays the text with strikethrough on the active tab.

ActiveTabFontUnderline Display the text with underlines on the active tab.

FontBold Display bold text on non-active tabs.

FontItalic Display italic text on non-active tabs.

FontName Specifies the font name for the text for all tabs.

FontSize The size of the text for all tabs.

FontStrikeThru Displays the text with strikethrough on non-active tabs.

FontUnderline Display the text with underlines on non-active tabs.

Tab Properties

Rows - [r]	Returns the number of rows of tabs.
Tab	Sets the active tab.
TabCutSize	Sets the size of the cut corners of each tab.
TabEnabled() - [r]	Determines whether the tab is enabled or not.
TabHwnd() - [r]	Returns the window handle of each tab.
TabOrientation	Sets whether the tabs will appear at the top, left, right or bottom of the control.
Tabs	Sets the total number of tabs.
TabRowOffset	Sets how much of tab row will be offset from the row in front of it.
TabsPerRow	Sets the number of tabs per row.
TabSelectType	Determines how the tabs are rearranged on the control when a new tab is selected.
TabStart	Determines which tab will be the active tab when the control loads.
TabVisible() - [r]	Determines whether the tab is visible or not.
TabHeight	Sets the height of all index tabs.
TabMaxWidth	Sets the maximum width of all index tabs.

Sizing/Location Properties

Align	Standard VB Align property.
Height	The height of the control.
Left	X coordinate of the left of the control.
Top	Y coordinate of the top of the control.
Width	The width of the control.

Picture Properties

ActiveTabPicture	Specifies the picture to use on the active index tab.
Picture	A designtime only property which specifies the picture for the currently selected index tab.
AlignmentPicture	Determines how the picture will be aligned for all tabs.
TabPictureMetaHeight	Sets the height of a metafile when placed on an index tab.
TabPictureMetaWidth	Sets the width of a metafile when placed on an index tab.
TabPicture() - [r]	Specifies the picture to display on a particular tab.

Other Properties

(About)	Displays version information about the control at design time.
DragIcon	The icon to use in a drag operation.
DragMode	The drag mode.
Enabled	Determines if the control is enabled.
Index	The index of the control if it is in a control array.
MousePointer	Determines the type of mouse pointer displayed.
Name	The name of the control.
Redraw	Disables the control's ability to repaint.
ShowFocusRect	Determines whether the focus rectangle is displayed on a tab when the tab has focus.
TabIndex	The standard Visual Basic TabIndex property.
TabStop	Determines if the user can get focus on the control by hitting tab.
Tag	The tag for the control.
Visible	Determines if the control is visible.

The FormFX Control

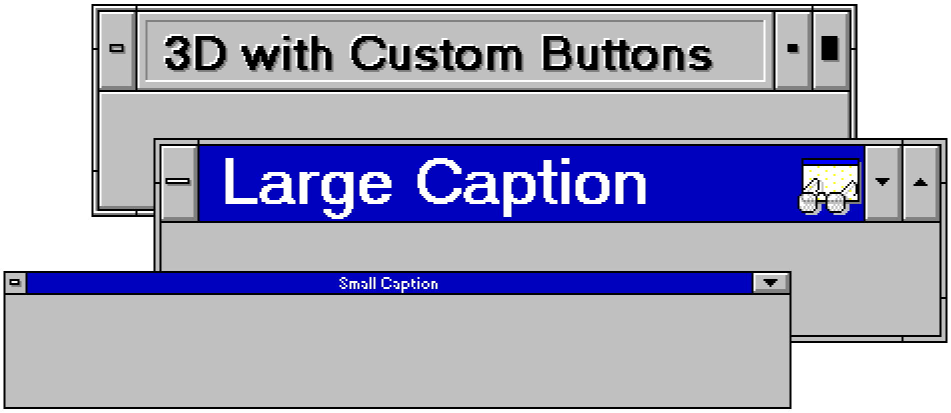
Properties

Description

FormFX is a custom control that allows you to customize the look and behavior your forms.

It gives you the ability to customize the size of the caption area and display multi-line text and pictures in the caption. In addition, you can add 3D effects to the caption, borders and/or client area.

There are also several properties that can be used to customize the behavior of a form such as restricting its size and movement.



File Name

SSFORMFX.VBX

ObjectType

SSFormFX

Related Topics:

Properties Quick Reference
Parts of a Form
Customizing the Caption Buttons
Notes on Setting the Caption of a Form
Control Limitations

Properties that apply only to this control are marked with an asterisk (*). For all other standard properties, see the *Microsoft Visual Basic Language Reference***.**

* (About)
* AlignmentCaption
* AlignmentPicture
* AlwaysOnTop
* BackColorActive
* BackColorInactive
* BevelColorFace
* BevelColorHighlight
* BevelColorShadow
* Border3D
* Caption
* Caption3D
* CaptionBevelInner
* CaptionBevelOuter
* CaptionBevelWidth
* CaptionBorderWidth
* CaptionButtonsPic
* CaptionHeight
* CaptionMultiLine
* CaptionPicture
* CaptionPictureMetaHeight
* CaptionPictureMetaWidth
* CaptionPictureX
* CaptionPictureY
* Client3D
* ClientBevelInner
* ClientBevelOuter
* ClientBevelWidth
* ClientBorderWidth
* ClientMove
* ControlBox3D
* ControlBoxWidth
* Font3D

FontBold
FontItalic
FontName
FontSize
FontStrikeThru
FontUnderline

* ForeColorActive
* ForeColorInactive

Height
Index

* InstantClose

Left

* LockCtlKeyOverride
* LockMove
* LockSize
* MaximizeHeight
* MaximizeLeft
* MaximizeTop
* MaximizeWidth

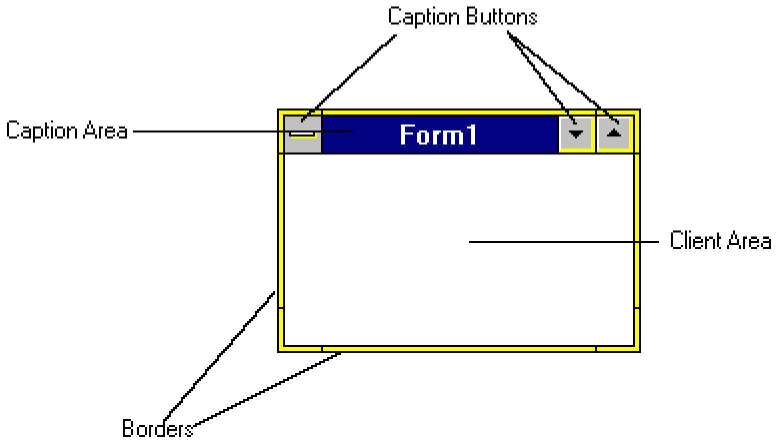
Name

* Redraw
* SizeMaxHeight
* SizeMaxWidth
* SizeMinHeight
* SizeMinWidth
* StdButtonHeight
* StdCaption
* StdCaptionHeight

Tag
Top
Width

Parts of a Form

The FormFX control can be used to customize different areas of Visual Basic forms. The diagram below illustrates the parts of a form:



A standard Visual Basic form is divided into 2 basic parts, the client area and the non-client area. The non-client area includes the borders, the caption area and the caption buttons (the control box, minimize, maximize and restore buttons). The client area is were controls are placed.

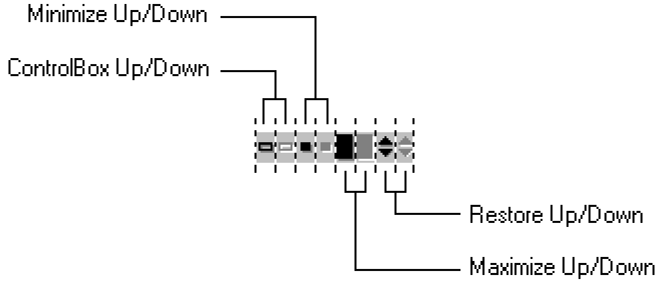
SSFormFX allows you to change the button height, the control box height, the caption height, the numbers of lines of caption text and the font used for the caption. Pictures can also be added to the caption area. You can also specify a bitmap to customize the look of the caption buttons. In addition, you can make the border appear 3D.

In addition, there are also several properties that can be used to customize the behavior of a form such as restricting its size and movement.

Customizing the Caption Buttons

FormFX lets you customize the buttons that appear in the caption (the control box, minimize, maximize and restore buttons). If you want to customize these buttons, you need to assign a bitmap to the **CaptionButtonsPic** property.

The **CaptionButtonsPic** property is used to specify a bitmap that contains both the up and down states for all four caption buttons. The bitmap is a standard Windows bitmap created by any bitmap editor (such as the standard PaintBrush application). It is broken up into 8 equal segments. Each segment represents the picture to use for a state (up or down) of one of the buttons. The first segment contains an image for the control box's up position, the second contains the image for its down position, and so on. Below is a sample bitmap (supplied with Designer Widgets in the SAMPLES directory) that illustrates the different segments:



The FormFX control uses this bitmap to display the buttons in the caption area. The control divides the bitmap evenly into 8 segments, then assigns each segment of the bitmap to each button. The segments do not have to be a specific width or size, as long as they are equal in size.

If a segment is smaller than a button, the FormFX control will center the image and fill the rest of the area with the color defined in the **BevelColorFace** property. If it is larger than the button, the control apply the center of the image over the button for as large as the button's dimensions.

Notes on Setting the Caption of a Form

Due to the way Visual Basic behaves when setting the Form's **Caption** property at runtime, unusual painting problems may occur.   If you need to set the **Caption** property at runtime, set the FormFX's **Caption** property instead.   The FormFX control will use the text in this property to display in the Form's caption area.   If you leave the FormFX's **Caption** property blank at design time, the FormFX control will use the text in the Form's **Caption** property.

Control Limitations

Although the FormFX provides you with extensive flexibility to customize forms in Visual Basic, there are some limitations you should be aware of when using the control.   The limitations are:

1. The form cannot have scrollbars.   If you are using a custom control to provide scroll bars on a form (such as VBAssist's Form Scroll Control), place a picture box on the form and size it so that it is the same size as the form's client area.   Then place the scroll control and all other control you want to scroll inside the picture box.
2. The form cannot be an MDI parent form or MDI Child form.
3. The form cannot have a menu.

If the SSFormFX control is used under any of the above circumstances, the properties used to customize the form will be ignored.

Properties Quick Reference

This section will give you a quick reference to all properties grouped into functional categories. For a complete reference of all properties, see Chapter 11.

Related Topics:

Caption Properties
3D Effects Properties
Font Properties
Color Properties
Caption Button Properties
Form Size/Movement Properties
Picture Properties
Other Properties

Caption Properties

Caption	Sets the text for the caption.
AlignmentCaption	Determines where the caption will be aligned in the caption area.
CaptionHeight	Sets the height of the caption area.
CaptionMultiLine	Determines whether the caption text can be more than one line.
StdCaption	Determines whether the standard caption will be displayed, ignoring all other properties relating to the caption area.

3D Effects Properties

BevelColorFace	Sets the color of the face of the bevel.
BevelColorHighlight	Sets the color of the bevel highlight.
BevelColorShadow	Sets the color of the bevel shadow.
Border3D	Determines whether the border of the form will have a bevel.
Caption3D	Determines whether the caption area will be 3D.
CaptionBevelInner	Specifies the type of inside bevel for the caption area.
CaptionBevelOuter	Specifies the type of outside bevel for the caption area.
CaptionBevelWidth	Sets the width of the bevel for the caption area.
CaptionBorderWidth	Sets the width of space between the inner and outer bevels.
Client3D	Determines whether the client area will have a 3D bevel around its edge.
ClientBevelInner	Specifies the type of inside bevel for the client area.
ClientBevelOuter	Specifies the type of outside bevel for the client area.
ClientBevelWidth	Sets the width of the bevel for the client area.
ClientBorderWidth	Sets the width of space between the inner and outer bevels.
ControlBox3D	Determines whether the control box will have beveled edges.
Font3D	Displays the text in 3-D.

Font Properties

FontBold Sets the bold attribute for the text.

FontItalic Sets the italic attribute for the text.

FontName Determines what type of font will be used.

FontSize Sets the font size.

FontStrikeThru Sets the strikethru attribute for the text.

FontUnderline Sets the underline attribute for the text.

Color Properties

BackColorActive Sets the background color of the caption area when the form is active.

BackColorInactive Sets the background color of the caption area when the form is active.

ForeColorActive Sets the foreground color for the caption area when the form is active.

ForeColorInactive Sets the foreground color for the caption area when the form is inactive.

Caption Button Properties

CaptionButtonsPic Specifies a single bitmap that contains an embedded picture for each button.

ControlBoxWidth Sets the width of the control box.

InstantClose Determines whether a single click on the control box will close the form.

StdButtonHeight Determines whether the buttons in the caption area will be standard height.

Form Size/Movement Properties

AlwaysOnTop	Specifies if the form will always be the top most window.
ClientMove	Determines whether the form can be moved by clicking on the client area.
LockCtlKeyOverride	Allows the CTRL key to override restricted movement or sizing.
LockMove	Determines whether the form can be moved.
LockSize	Determines whether the size of the form can be changed by the user.
MaximizeHeight	Specifies the height of the form when maximized.
MaximizeLeft	Specifies the x-coordinate of the form when maximized.
MaximizeTop	Specifies the y-coordinate of the form when maximized.
MaximizeWidth	Specifies the width of the form when maximized.
SizeMaxHeight	Specifies the maximum height the form can be sized to.
SizeMaxWidth	Specifies the maximum width the form can be sized to.
SizeMinHeight	Specifies the minimum height the form can be sized to.
SizeMinWidth	Specifies the minimum width the form can be sized to.

Picture Properties

AlignmentPicture Determines where the picture will appear in the caption area.

CaptionPicture Specifies the picture for the caption area.

CaptionPictureMetaHeight Sets the height of a metafile when placed in the caption area.

CaptionPictureMetaWidth Sets the width of a metafile when placed caption area.

CaptionPictureX Determines the x-position in the caption area where to draw the picture.

CaptionPictureY Determines the y-position in the caption area where to draw the picture.

Other Properties

(About)	Displays version information about the control at design time.
Height	The height of the control.
Index	The index of the control if it is in a control array.
Left	The x coordinate of the control.
Name	The name of the control.
Redraw	Disables the control's ability to repaint.
Tag	The tag for the control.
Top	The Y coordinate of the control.
Width	The width of the control.

Properties Reference

The following is a complete reference of all the custom properties in the Designer Widgets custom controls. For all other standard properties, see the *Microsoft Visual Basic Language Reference***.**

Related Topics:

(About) Property
(ToolbarDesigner) Property
ActiveTabBackColor, ActiveTabForeColor Properties
ActiveTabFontBold, ActiveTabFontItalic, ActiveTabFontStrikeThru, ActiveTabFontUnderline Properties
ActiveTabPicture Property
AlignmentCaption Property
AlignmentPicture Property
AllowToolbarDockLeft, AllowToolbarDockRight, AllowToolbarDockTop, AllowToolbarDockBottom Properties
AllowToolbarFloat Property
AllowToolbarMove Property
AlwaysOnTop Property
BackColorActive, BackColorInactive Properties
BackColorDock Property
BackColorHelp, ForeColorHelp Properties
BalloonHelp Property
BalloonHelpDelay Property
BevelColorFace, BevelColorHighlight, BevelColorShadow Properties
BevelInner Property
BevelOuter Property
BevelWidth Property
Border3D Property
BorderWidth Property
BtnEnabled() Property
BtnGroupSpacing Property
BtnHeight Property
BtnMarginX BtnMarginY Properties
Btns Property
BtnSpacing Property
BtnToolNum()Property
BtnValue() Property
BtnVisible() Property
BtnWidth Property
Caption Property
Caption3D Property
CaptionBevelInner Property
CaptionBevelOuter Property
CaptionBevelWidth Property
CaptionBorderWidth Property
CaptionButtonsPic Property
CaptionHeight Property
CaptionMultiLine Property
CaptionPicture Property
CaptionPictureMetaHeight, CaptionPictureMetaWidth Properties
CaptionPictureX, CaptionPictureY Properties
Client3D Property
ClientBevelInner Property
ClientBevelOuter Property
ClientBevelWidth Property

ClientBorderWidth Property
ClientMove Property
ClientWidth Property
ClientHeight Property
ClientLeft Property
ClientTop Property
ControlBox3D Property
ControlBoxWidth Property
DockRank, DockRankSeq Properties
DockStatus Property
FloatingCaption Property
FloatingCaptionType Property
FloatingControlBox Property
FloatingLeft, FloatingTop Properties
FloatingWidth, FloatingHeight Properties
FloatingSizable Property
FloatingWidthInBtns Property
Font3D Property
ForeColorActive, ForeColorInactive Properties
HwndToolbar Property
InstantClose Property
LockCtlKeyOverride Property
LockMove Property
LockSize Property
MaximizeHeight, MaximizeWidth Properties
MaximizeLeft, MaximizeTop Properties
Outline Property
Picture Property
Redraw Property
Rows Property
ShowFocusRect Property
SizeMaxHeight, SizeMaxWidth Properties
SizeMinHeight, SizeMinWidth Properties
StdButtonHeight Property
StdCaption Property
StdCaptionHeight Property
Tab Property
TabBackColor(), TabForeColor() Properties
TabCaption() Property
TabCutSize Property
TabEnabled() Property
TabHeight Property
TabHwnd() Property
TabMaxWidth Property
TabOrientation Property
TabPicture() Property
TabPictureMetaHeight, TabPictureMetaWidth Properties
TabRowOffset Property
Tabs Property
Tabs3D Property
TabSelectType Property
TabsPerRow Property
TabStart Property
TabVisible() Property
Toolbar3D Property
ToolsetName Property

ToolsetNumBtnStates Property
ToolsetNumTools Property
ToolsetPicture Property
ToolsetToolAllowAllUp() Property
ToolsetToolDesc() Property
ToolsetToolExclusive() Property
ToolsetToolGroup() Property
ToolsetToolHelp() Property
ToolsetToolID() Property
ToolsetToolMnemonic() Property
ToolsetToolPicDn(), ToolsetToolPicUp() Properties
ToolsetToolPicDnDis(), ToolsetToolPicUpDis() Properties
ToolsetToolType() Property

(About) Property

Applies To

SSToolbar, SSFormFX, SSIndexTab

Description

Displays version information about the control.

Usage

Click on the ellipses ('...') button next to the property text to activate the about dialog box.

Remarks

Available only at design time.

(ToolbarDesigner) Property

Applies To

SSToolbar

Description

This property lets the developer invoke the Toolbar Designer in order to create and maintain toolbars and toolsets at design time.

Usage

Click on the ellipses ('...') button next to the property text to activate the Toolbar Designer.

Remarks

Refer to Chapter 8 - 'The Dockable Toolbar' under the section heading 'Using the Toolbar Designer' for a detailed explanation of this tool.

Available only at design time.

ActiveTabBackColor, ActiveTabForeColor Properties

Applies To

SSIndexTab

Description

Sets the background and/or foreground color of the active index tab.

Usage

[*form.*]*control.***ActiveTabBackColor**[= *color*]

[*form.*]*control.***ActiveTabForeColor**[= *color*]

Remarks

These properties set the color for background and/or foreground on the active index tab. If the **Tabs3D** property is set to **True**, the **ActiveTabBackColor** property is ignored and **BevelColorFace** is used. Otherwise, the **ActiveTabBackColor** property is used for the active tab and the entire client area of the tab.

Data Type

Long

See Also

ActiveTabFontBold, ActiveTabFontItalic, ActiveTabFontStrikeThru, ActiveTabFontUnderline, ActiveTabPicture, Tabs3D, TabBackColor(), TabForeColor()

ActiveTabFontBold, ActiveTabFontItalic, ActiveTabFontStrikeThru, ActiveTabFontUnderline Properties

Applies To

SSIndexTab

Description

Sets the font attributes for the text on the active index tab.

Usage

[*form.*]*control.***ActiveTabFontBold**[= {**True**|**False**}]

[*form.*]*control.***ActiveTabFontItalic**[= {**True**|**False**}]

[*form.*]*control.***ActiveTabFontStrikeThru**[= {**True**|**False**}]

[*form.*]*control.***ActiveTabFontUnderline**[= {**True**|**False**}]

Remarks

These properties determine the font attributes of the text that appears on the active tab.

The property settings are:

Setting	Description
True	Enables the font attribute.
False	(Default) Disables the font attribute.

Data Type

Integer (Boolean)

See Also

ActiveTabBackColor, **ActiveTabForeColor**, **ActiveTabPicture**

ActiveTabPicture Property

Applies To

SSIndexTab

Description

Specifies the picture to use on the active index tab.

Usage

[*form.*]*control.***ActiveTabPicture**[= *picture*]

Remarks

This property specifies the picture that will always be displayed on the active index tab. This picture specified in this property will override the picture set in the **TabPicture()** property.

Data Type

Integer (Picture)

See Also

ActiveTabBackColor, **ActiveTabForeColor**

AlignmentCaption Property

Applies To

SSFormFX, SSIndexTab

Description

Determines how the caption will be aligned for tabs or forms.

Usage

[*form.*]*control.***AlignmentCaption**[= s*etting*]

Remarks

The settings for this property are:

Setting	Description
0	Left Justify - Top
1	Left Justify - Middle
2	Left Justify - Bottom
3	Right Justify - Top
4	Right Justify - Middle
5	Right Justify - Bottom
6	Center - Top
7	(Default) Center - Middle
8	Center - Bottom

Data Type

Integer (Enumerated)

See Also

AlignmentPicture

Example

The following sample aligns the text to the left and aligns the   picture to the left of the text:

```
SSFormFX.AlignmentPicture = 9  'Left of Text
SSFormFX.AlignmentCaption = 1  'Left Justify - MIDDLE
```

AlignmentPicture Property

Applies To

SSFormFX, SSIndexTab

Description

Determines the how the picture will be aligned.

Usage

[*form.*]*control.***AlignmentPicture**[= *setting*]

Remarks

The settings for this property are:

Setting	Description
0	Left Justify - Top
1	(Default) Left Justify - Middle
2	Left Justify - Right
3	Right Justify - Top
4	Right Justify - Middle
5	Right Justify - Right
6	Center - Top
7	Center - Middle
8	Center - Right
9	Left of Text
10	Right of Text
11	Above Text
12	Below Text
13	Fit to Tab/Caption
The follow options are only available with SSFormFX:	
14	Fixed from Left - The picture will be positioned on the caption using the **CaptionPictureX** and **CaptionPictureY** properties relative to the top left of the caption.
15	Fixed from Right - The picture will be positioned on the caption using the **CaptionPictureX** and **CaptionPictureY** properties relative to the top right of the caption.

Data Type

Integer (Enumerated)

See Also

Picture, **CaptionPictureX**, **CaptionPictureY**

Example

The following sample aligns the text to the left and aligns a picture to the left of the text:

```
SSFormFX.AlignmentPicture = 9  'Left of Text
SSFormFX.AlignmentCaption = 1  'Left Justify - MIDDLE
```

AllowToolbarDockLeft, AllowToolbarDockRight, AllowToolbarDockTop, AllowToolbarDockBottom Properties

Applies To

SSToolbar

Description

Determines where a toolbar is allowed to dock at runtime.

Usage

[*form.*]*control.***AllowToolbarDockLeft**[= {**True**|**False**}]

[*form.*]*control.***AllowToolbarDockRight**[= {**True**|**False**}]

[*form.*]*control.***AllowToolbarDockTop**[= {**True**|**False**}]

[*form.*]*control.***AllowToolbarDockBottom**[= {**True**|**False**}]

Remarks

The property settings are:

Setting	Description
True	(Default) The toolbar can dock in this area.
False	The toolbar cannot dock in this area.

These properties only have meaning for SSToolbar controls that are placed directly on MDI forms (not on containers).

The docking areas are outside of the MDI client area so they do not interfere with MDI child forms.

Data Type

Integer (Boolean)

See Also

DockRank, **DockRankSeq**, **DockStatus**

Example

This sample code restricts the user from docking the toolbar in the left and right docking areas:

```
SSToolbar1.AllowToolbarDockLeft  = False
SSToolbar1.AllowToolbarDockRight = False
```

AllowToolbarFloat Property

Applies To

SSToolbar

Description

Determines if a toolbar is allowed to float at runtime.

Usage

[*form.*]*control.***AllowToolbarFloat**[= {**True**|**False**}]

Remarks

The property settings are:

Setting	Description
True	(Default) The toolbar can be dragged off a docking area.
False	The toolbar cannot float.

The **AllowToolbarFloat** property controls whether the toolbar can be dragged off a docking area and remain as a floating toolbar.

This property only has meaning for SSToolbar controls that are placed directly on MDI forms (not on containers).

Data Type

Integer (Boolean)

See Also

AllowToolbarMove

AllowToolbarMove Property

Applies To

SSToolbar

Description

Determines if a toolbar is allowed to move at runtime.

Usage

[*form.*]*control.***AllowToolbarMove**[= {**True**|**False**}]

Remarks

The property settings are:

Setting	Description
True	(Default) The toolbar can moved.
False	The toolbar cannot be moved.

This property, if set to **False,** is used to create a "fixed" toolbar which cannot be moved off a docking area at runtime. When a toolbar is fixed, it is initially placed onto the docking area determined by the **DockStatus** property setting.

This property only has meaning for SSToolbar controls that are placed directly on MDI forms (not on containers).

Data Type

Integer (Boolean)

See Also

AllowToolbarFloat

AlwaysOnTop Property

Applies To

SSFormFX

Description

Specifies if the form will always remain as the top most window.

Usage

[*form.*]*control.***AlwaysOnTop**[= {**True**|**False**}]

Remarks

When a form is the top most window, it remains at the top of the Z-order even when it is deactivated.

The property settings are:

Setting	Description
True	The form will always be the top most window.
False	(Default) The form will only be the top most window when it is the active window.

Data Type

Integer (Boolean)

BackColorActive, BackColorInactive Properties

Applies To

SSFormFX

Description

Sets the background color of the caption area of the form when it is the active or inactive window.

Usage

[*form.*]*control.***BackColorActive**[= *color*]

[*form.*]*control.***BackColorInactive**[= *color*]

Remarks

These properties specify the background color of the caption area when the form is active or inactive. If the **Caption3D** property is **True**, these properties are ignored and the **BevelColorFace** property is used. These properties default to the color set in the **Windows Control Panel**.

The **StdCaption** property must be set to **False** for this property to take effect.

Data Type

Long

See Also

ForeColorActive, **ForeColorInactive**

BackColorDock Property

Applies To

SSToolbar

Description

Sets the background color of all docking areas.

Usage

[*form.*]*control.***BackColorDock**[= *color*]

Remarks

The docking areas are only used for SSToolbar controls that are placed directly on MDI forms (not on containers).

Data Type

Long

See Also

AllowToolbarDockLeft, **AllowToolbarDockTop**, **AllowToolbarDockRight**, **AllowToolbarDockBottom**, **DockStatus**

BackColorHelp, ForeColorHelp Properties

Applies To

SSToolbar

Description

Sets the background color and foreground (text) color for balloon help.

Usage

[*form.*]*control.***BackColorHelp**[= *color*]

[*form.*]*control.***ForeColorHelp**[= *color*]

Remarks

A balloon help window is displayed when the user pauses over a button that has its **ToolsetToolHelp()** property set.

Data Type

Long

See Also

BalloonHelp, **BalloonHelpDelay**, **ToolsetToolHelp**

BalloonHelp Property

Applies To

SSToolbar

Description

Determines if balloon help will appear when the mouse pointer is paused over a tool on the toolbar.

Usage

[*form.*]*control.***BalloonHelp**[= {**True**|**False**}]

Remarks

When balloon help is activated, a 'balloon help' window will be displayed just below the cursor. This window will contain the text that was set in the tool's **ToolsetToolHelp()** property. The balloon help window is only displayed after the mouse stops moving over the tool for the duration specified in the **BalloonHelpDelay** property.

The property settings are:

Setting	Description
True	(Default) A balloon help window will appear when the mouse pointer is paused long enough (based on the **BalloonHelpDelay** property) over a tool that has text in its **ToolsetToolHelp()** property.
False	A balloon help window will not appear.

Data Type

Integer (Boolean)

See Also

BackColorHelp, **BalloonHelpDelay**, **ForeColorHelp**, **ToolsetToolHelp()**

Example

This sample code sets the toolbar to activate balloon help when the mouse is over a button for over 1 second:

```
SSToolbar1.BalloonHelpDelay = 1000  'Set delay to 1 second
SSToolbar1.BalloonHelp = True       'Activate balloon help
```

BalloonHelpDelay Property

Applies To

SSToolbar

Description

Sets or returns the amount of time (in milliseconds) that the mouse has to pause over a tool before balloon help window appears.

Usage

[*form.*]*control.***BalloonHelpDelay**[= *milliseconds*]

Remarks

This property specifies the amount of time in milliseconds to wait before displaying the balloon help window while the mouse pointer is paused over a tool.

The help window will not appear for the following reasons:

- The mousepointer does not pause over the tool long enough.
- The **BalloonHelp** property is set to **False.**
- There is no text in the tool's **ToolsetToolHelp()** property.

The range of the delay is 0 to 5000 milliseconds (1 second equals 1000 milliseconds). The default is 500 or half a second.

Data Type

Long

See Also

BackColorHelp, **BalloonHelp**, **ForeColorHelp**, **ToolsetToolHelp()**

Example

This sample code sets the toolbar to activate balloon help when the mouse is over a button for over 1 second:

```
SSToolbar1.BalloonHelpDelay = 1000  'Set delay to 1 second
SSToolbar1.BalloonHelp = True       'Activate balloon help
```

BevelColorFace, BevelColorHighlight, BevelColorShadow Properties

Applies To

SSFormFX, SSIndexTab, SSToolbar

Description

Sets the colors used to draw 3D bevels.

Usage

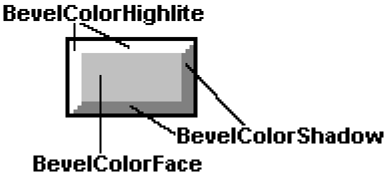
[*form.*]*control.***BevelColorFace**[= *color*]

[*form.*]*control.***BevelColorHighlight**[= *color*]

[*form.*]*control.***BevelColorShadow**[= *color*]

Remarks

When the control needs to draw a bevel, it uses the colors specified in these properties. These properties refer to the different parts of the bevels as shown in the diagram.



Data Type

Long

See Also

Caption3D, **Client3D**, **Tabs3D**, **Toolbar3D**

BevelInner Property

Applies To

SSToolbar

Description

Specifies the type of inside beveling for the toolbar.

Usage

[*form.*]*control.***BevelInner**[= s*etting*]

Remarks

This property is used to determine the type of inside beveling to draw on the toolbar. This property is ignored if the **Toolbar3D** property is **False.**

The settings for this property are:

Setting	Description
0	(Default) None. No inner bevel is drawn.
1	Inset. The inner bevel appears as if it is inset into the screen.
2	Raised. The inner bevel appears as if it is raised from the screen.

Data Type

Integer (Enumerated)

See Also

Toolbar3D, **BevelOuter**, **BevelColorFace**, **BevelColorHighlight**, **BevelColorShadow**

BevelOuter Property

Applies To

SSToolbar

Description

Specifies the type of outside beveling for the toolbar.

Usage

[*form.*]*control.***BevelOuter**[= *setting*]

Remarks

This property is used to determine the type of outside beveling to draw on the toolbar.   This property is ignored if the **Toolbar3D** property is **False.**

The settings for this property are:

Setting	Description
0	None. No outer bevel is drawn.
1	Inset. The outer bevel appears as if it is inset into the screen.
2	(Default) Raised. The outer bevel appears as if it is raised from the screen.

Data Type

Integer (Enumerated)

See Also

Toolbar3D, **BevelInner**, **BevelColorFace**, **BevelColorHighlight**, **BevelColorShadow**

BevelWidth Property

Applies To

SSToolbar

Description

Sets or returns the width of both inner and outer bevels of the toolbar, which determines the amount of the 3-D shadow effect.

Usage

[*form.*]*control.***BevelWidth**[= *width*]

Remarks

This property determines the number of pixels used to draw the inner and outer bevels which surround the toolbar. The valid range for this property is 0 to 30.   This property is ignored if the **Toolbar3D** property is **False**.

Data Type

Integer

See Also

BevelInner, **BevelOuter**

Border3D Property

Applies To

SSFormFX

Description

Determines whether the border of the form will be displayed in 3D.

Usage

[*form.*]*control.***Border3D**[= {**True**|**False**}]

Remarks

The property settings are:

Setting	Description
True	The border of the form will be beveled.
False	(Default) The border of the form will not be beveled.

Data Type

Integer (Boolean)

See Also

Caption3D, **Client3D**

BorderWidth Property

Applies To

SSToolbar

Description

Sets or returns the width of the space between the outer and inner bevels.

Usage

[*form.*]*control.***BorderWidth**[= *width*]

Remarks

This property determines the width in pixels between the inner and outer bevels which surround the toolbar. The valid range for this property is 0 to 30.

Data Type

Integer

See Also

BevelInner, **BevelOuter**

BtnEnabled() Property

Applies To

SSToolbar

Description

Enables or disables a button.

Usage

[*form.*]*control.***BtnEnabled**(*button*)[= {**True**|**False**}]

Remarks

When a button is disabled, it is redrawn using the button image supplied in the **ToolsetPicture** bitmap for the disabled state and the current value (up or down).

The property settings are:

Setting	Description
True	(Default) Button will be enabled.
False	Button will be disabled.

This property is only available at runtime.   However, it can be initially set at design time using the Toolbar Designer.

Data Type

Integer (Boolean)

See Also

BtnVisible()

Example

This sample code shows a function that toggles a buttons enabled state and returns the new state:

```
Function DisableButton(BtnNumber As Integer) As Integer

   SSToolbar1.BtnEnabled(BtnNumber) = Not SSToolbar1.BtnEnabled(BtnNumber)
   DisableButton = SSToolbar1.BtnEnabled(BtnNumber)

End Function
```

BtnGroupSpacing Property

Applies To

SSToolbar

Description

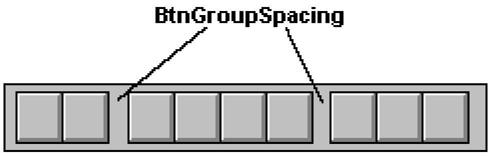
Sets or returns the number of pixels between button groups.

Usage

[*form.*]*control.***BtnGroupSpacing**[= *spacing*]

Remarks

This property determines the number of pixels between button groups.   This value is applied to all button groups within a toolbar.



Data Type

Integer

See Also

BtnSpacing, **ToolsetToolGroup()**

Example

This sample code sets the space between buttons to be -1 pixels (to cause a 1 pixel overlap) and the group spacing to be 8 pixels:

```
SSToolbar1.BtnSpacing      = -1  'causes a 1 pixel overlap
SSToolbar1.BtnGroupSpacing = 8   'set to 8 pixels
```

BtnHeight Property

Applies To

SSToolbar

Description

Returns the height (in pixels) of the buttons on the toolbar.

Usage

[*form.*]*control.***BtnHeight**[= *height*]

Remarks

The control will calculate this value by dividing the height of the entire bitmap specified in the **ToolsetPicture** property by the number of button states specified in the **ToolsetNumBtnStates** property.

This property is only available at runtime and is read-only.

Data Type

Integer

See Also

BtnWidth

BtnMarginX BtnMarginY Properties

Applies To

SSToolbar

Description

Specifies the distance (in pixels)   between the toolbar buttons and the inside bevels of the toolbar. If the toolbar has no bevels (**Toolbar3D** = **False**) then the distance is calculated from the left and top of the toolbar respectively.

Usage

[*form.*]*control.***BtnMarginX**[= *distance from left*]

[*form.*]*control.***BtnMarginY**[= *distance from top*]

Remarks

This property is available in both design and run modes.

Data Type

Integer

See Also

BtnGroupSpacing, **BtnSpacing**

Btns Property

Applies To

SSToolbar

Description

Specifies the number of buttons on the toolbar.

Usage

[*form.*]*control*.**Btns**[= *Btns*]

Remarks

This property is set automatically when a toolbar is defined using the Toolbar Designer. However, you may set this property to increase or decrease the number of buttons on the toolbar. If you increase the number of buttons, you must supply a value for the **BtnToolNum()** property of each new button. If the **BtnToolNum()** property is not set for a button, it will be -1 and the following bitmap is displayed instead:

This property is only available at runtime.

Data Type

Integer

See Also

BtnToolNum()

Example

The following code adds a button to the toolbar and assigns a 'Cut' tool from a predefined toolset:

```
SSToolbar1.Btns = SSToolbar1.Btns + 1 'create the new button
SSToolbar1.BtnToolNum(SSToolbar1.Btns - 1) = 2 'Assign the
                                               'Cut tool
```

BtnSpacing Property

Applies To

SSToolbar

Description

Sets or returns how much space will separate each button with the same group number.

Usage

[*form.*]*control.***BtnSpacing**[= *spacing*]

Remarks

This property represents the number of pixels between buttons in the same group.   Setting this to -1 will draw the buttons overlapped by one pixel so that only one border will appear between each button.   The valid range for this property is -1 to 30.

Data Type

Integer

See Also

BtnGroupSpacing, ToolsetToolGroup()

Example

This sample code sets the space between buttons to be -1 pixels (to cause a 1 pixel overlap) and the group spacing to 8 pixels:

```
SSToolbar1.BtnSpacing      = -1  'causes a 1 pixel overlap
SSToolbar1.BtnGroupSpacing = 8   'set to 8 pixels
```

BtnToolNum()Property

Applies To

SSToolbar

Description

Sets or returns the tool index (within the toolbar's toolset) that is associated with the button.

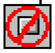
Usage

[*form.*]*control.***BtnToolNum**(*button*)[= *toolnum*]

Remarks

This property is set automatically when a toolbar is defined using the Toolbar Designer. However, you can set this property at runtime to a valid tool number of a tool in the current toolset.

This property is initialized to -1. If this property is not defined for a button, the following bitmap is displayed instead:



This property is only available at runtime.

Data Type

Integer

See Also

ToolsetToolID()

Example

The following code adds a button to the toolbar and assigns a 'Cut' tool from a predefined toolset:

```
SSToolbar1.Btns = SSToolbar1.Btns + 1 'create the new button
SSToolbar1.BtnToolNum(SSToolbar1.Btns - 1) = 2 'Assign the
                                               'Cut tool
```

BtnValue() Property

Applies To

SSToolbar

Description

Sets or returns the button's value.

Usage

[*form.*]*control.***BtnValue**(*button*)[= {**True**|**False**}]

Remarks

The property settings are:

Setting	Description
True	The button is in the down position.
False	(Default) The button is in the up position.

This property is only available at runtime.   However, it can be initially set at design time using the Toolbar Designer.

Data Type

Integer (Boolean)

See Also

ToolsetToolType()

Example

This sample checks the **BtnValue()** property to see if a button is pressed when the mouse is moved over a button:

```
Sub SSToolbar1_MouseEnter(ToolID As String, ToolNum As Integer, Btn As Integer)
   If SSToolbar1.BtnValue(Btn) Then
       MsgBox "The Button is Pressed!"
   Else
       MsgBox "The Button is NOT Pressed!"
   End If
End Sub
```

BtnVisible() Property

Applies To

SSToolbar

Description

Determines whether the button will be visible.

Usage

[*form.*]*control.***BtnVisible**(*button*)[= {**True**|**False**}]

Remarks

The property settings are:

Setting	Description
True	(Default) Button will be visible.
False	Button will not be visible.

This property is only available at runtime.   However, it can be initially set at design time using the Toolbar Designer.

Data Type

Integer (Boolean)

See Also

BtnEnabled()

Example

This sample code hides a button:

```
SSToolbar1.BtnVisible(0) = False
```

BtnWidth Property

Applies To

SSToolbar

Description

Returns the width of all buttons in a toolbar.

Usage

[*form.*]*control.***BtnWidth**[= *width*]

Remarks

The control will calculate this value by dividing the width of the entire bitmap specified in the **ToolsetPicture** property by the number of tools specified in the **ToolsetNumTools** property.

This property is only available at runtime and is read-only.

Data Type

Integer

See Also

BtnHeight

Caption Property

Applies To

SSFormFX, SSIndexTab

Description

Specifies the caption for the form, or an individual tab at design time.

Usage

[*form.*]*control.***Caption**[= *text*]

Remarks

When used with SSFormFX, this property sets the caption of the form.

Note If you want to set the caption of the form at runtime, use this property and not the form's caption property. However, if you leave this property blank, it will default to the form's caption.

When used with the SSIndexTab at design time, this property sets the **TabCaption()** property of the current tab specified in the **Tab** property.

Data Type

String

See Also

AlignmentCaption, **CaptionMultiLine**

Example

This sample code sets the caption height in the SSFormFX control to be 2000 twips to fit a multiline caption:

```
SSFormFX1.CaptionHeight = 2000  'Set CaptionHeight=2000 twips
SSFormFX1.CaptionMultiLine = True
SSFormFX1.Caption = "This is a" + Chr$(13) + Chr$(10) + "Multiline Caption"
```

Caption3D Property

Applies To

SSFormFX

Description

Determines whether the caption area of the form will be 3D.

Usage

[*form.*]*control.***Caption3D**[= {**True**|**False**}]

Remarks

The property settings are:

Setting	Description
True	The caption area will be displayed in 3D.
False	(Default) The caption area will not be 3D.

The **StdCaption** property must be set to **False** for this property to take effect.

Data Type

Integer (Boolean)

See Also

CaptionBevelInner, **CaptionBevelOuter**, **CaptionBevelWidth**, **CaptionBorderWidth**

CaptionBevelInner Property

Applies To

SSFormFX

Description

Specifies the type of inside bevel for the caption area of the form.

Usage

[*form.*]*control.***CaptionBevelInner**[= *setting*]

Remarks

This property is used to determine the type of inside beveling to draw in the caption area of the form. This property is ignored if the **Caption3D** property is **False.**

The settings for this property are:

Setting	Description
0	(Default) None. No inner bevel is drawn.
1	Inset. The inner bevel appears as if it is inset into the screen.
2	Raised. The inner bevel appears as if it is raised from the screen.

The **StdCaption** property must be set to **False** for this property to take effect.

Data Type

Integer (Enumerated)

See Also

Caption3D, **BevelColorFace**, **BevelColorHighlight**, **BevelColorShadow**, **CaptionBevelWidth**

CaptionBevelOuter Property

Applies To

SSFormFX

Description

Specifies the type of outside bevel for the caption area of the form.

Usage

[*form.*]*control.***CaptionBevelOuter**[= s*etting*]

Remarks

This property is used to determine the type of outside beveling to draw in the caption area of the form. This property is ignored if the **Caption3D** property is **False.**

The property settings are:

Setting	Description
0	None. No outer bevel is drawn.
1	Inset. The outer bevel appears as if it is inset into the screen.
2	(Default) Raised. The outer bevel appears as if it is raised from the screen.

The **StdCaption** property must be set to **False** for this property to take effect.

Data Type

Integer (Enumerated)

See Also

Caption3D, **BevelColorFace**, **BevelColorHighlight**, **BevelColorShadow**, **CaptionBevelWidth**

CaptionBevelWidth Property

Applies To

SSFormFX

Description

Sets or returns the width of the bevel (outer or inner) of the caption area of the form, which determines the amount of the 3-D shadow effect.

Usage

[*form.*]*control.***CaptionBevelWidth**[= *width*]

Remarks

This property determines the number of pixels used to draw the inner and outer bevels which surround the caption area of the form. The valid range for this property is 0 to 30. This property is ignored if the **Caption3D** property is **False**.

The **StdCaption** property must be set to **False** for this property to take effect.

Data Type

Integer

See Also

Caption3D, **CaptionBevelInner**, **CaptionBevelOuter**

CaptionBorderWidth Property

Applies To

SSFormFX

Description

Sets or returns the width of the space between the outer and inner bevels in the caption area of the form.

Usage

[*form.*]*control.***CaptionBorderWidth**[= *width*]

Remarks

This property determines the width in pixels between the inner and outer bevels which surround the caption area of the form. The valid range for this property is 0 to 30. This property is ignored if the **Caption3D** property is **False**.

The **StdCaption** property must be set to **False** for this property to take effect.

Data Type

Integer

See Also

Caption3D, **CaptionBevelInner**, **CaptionBevelOuter**

CaptionButtonsPic Property

Applies To

SSFormFX

Description

Specifies a bitmap containing the pictures to use for each button in the caption.

Usage

[*form.*]*control.***CaptionButtonsPic**[= *picture*]

Remarks

This property contains a single bitmap that contains 8 equal segments. Each segment contains the picture (excluding any bevels) for each of the buttons on a form, and for its up and down state. The first two segments of the bitmap represent the control box up and down, respectively, followed by an up and down picture for the minimize button, the maximize button and the restore button. A sample bitmap is included with the control in the SAMPLES directory and it is called **BUTTONS.BMP.**

Sample CaptionButtonsPic - BUTTONS.BMP

Data Type

Integer (Picture)

See Also

StdButtonHeight

Example

This sample codes sets the **CaptionButtonsPic** property to a bitmap:

```
SSFormFX1.CaptionButtonsPic = LoadPicture("BUTTONS.BMP")
```

CaptionHeight Property

Applies To

SSFormFX

Description

Sets or returns the height of the caption area of a form.

Usage

[*form.*]*control.***CaptionHeight**[= *height*]

Remarks

This property specifies the height of the caption area in the scale mode of the form. If you set this property to zero (0), the control will calculate the height of the caption based on the **FontSize** property.

If the **StdCaptionHeight** property is set to **True**, this property is ignored.

The **StdCaption** property must be set to **False** for this property to take effect.

Data Type

Single

See Also

StdCaptionHeight, **StdButtonHeight**

Example

This sample code sets the caption height to be 2000 twips to fit a multiline caption:

```
SSFormFX1.CaptionHeight = 2000  'Set CaptionHeight=2000 twips
SSFormFX1.CaptionMultiLine = True
SSFormFX1.Caption = "This is a" + Chr$(13) + Chr$(10) + "Multiline Caption"
```

CaptionMultiLine Property

Applies To

SSFormFX

Description

Determines whether the caption text on a form can be more than one line.

Usage

[*form.*]*control.***CaptionMultiLine**[= {**True**|**False**}]

Remarks

Setting this property to **True** will cause a long caption to wrap to the next line.   You can also break up a line into multiple lines by separating the caption text with a carriage return/line feed sequence (Chr$(13) + Chr$(10)).

The property settings are:

Setting	Description
True	The caption can be more than one line.
False	(Default) The caption can only be a single line.

The **StdCaption** property must be set to **False** for this property to take effect.

Data Type

Integer (Boolean)

See Also

Caption, **CaptionHeight**

Example

This sample code sets the caption height to be 2000 twips to fit a multiline caption:

```
SSFormFX1.CaptionHeight = 2000  'Set CaptionHeight=2000 twips
SSFormFX1.CaptionMultiLine = True
SSFormFX1.Caption = "This is a" + Chr$(13) + Chr$(10) + "Multiline Caption"
```

CaptionPicture Property

Applies To

SSFormFX

Description

Sets the picture to be drawn in the caption area on the form.

Usage

[*form.*]*control.***CaptionPicture**[= *picture*]

Remarks

This property specifies the picture to be displayed in the caption area. Valid pictures are bitmaps, icons and Windows metafiles. If you specify a Windows metafile, you must also set the height and width to display the picture via the **CaptionPictureMetaHeight** and **CaptionPictureMetaWidth** properties.

Data Type

Integer (Picture)

See Also

AlignmentPicture, CaptionPictureMetaHeight, CaptionPictureMetaWidth

Example

This sample code puts an icon left justified in the caption area:

```
SSFormFX1.CaptionHeight = 2000  'Make the caption big enough
SSFormFX1.CaptionPicture = LoadPicture("GOGGLES.ICO")
SSFormFX1.AlignmentPicture = 1 'left justify
```

CaptionPictureMetaHeight, CaptionPictureMetaWidth Properties

Applies To

SSFormFX

Description

Sets the height and/or width of the metafile if specified in the **CaptionPicture** property.

Usage

[*form.*]*control.***CaptionPictureMetaHeight**[= *height*]

[*form.*]*control.***CaptionPictureMetaWidth**[= *width*]

Remarks

These properties represent the height and/or width of the metafile when drawn in the caption area.

The units specified are based on the scale mode of the form.

Data Type

Single

See Also

CaptionPicture

CaptionPictureX, CaptionPictureY Properties

Applies To

SSFormFX

Description

Determines the caption picture's x/y-position relative to the left or right (depending upon the setting of the **AlignmentPicture** property) and top of the caption area.

Usage

[*form.*]*control.***CaptionPictureX** [= *x-coordinate*]

[*form.*]*control.***CaptionPictureY** [= y-*coordinate*]

Remarks

These properties are used in conjunction with the **AlignmentPicture** property to determine the fixed positions of the picture in the caption area. These properties are only used when **AlignmentPicture** is set to **'**Fixed from Right' or 'Fixed from Left'.

The **CaptionPictureY** property specifies the distance from the top of the caption area.

The **CaptionPictureX** property specifies the distance from either the left or right of the caption area. The X-coordinate is relative to the left of the caption area if the **AlignmentPicture** property is set to 'Fixed from Left' and relative to the right if 'Fixed from Right'.

The units specified are based on the scale mode of the form.

Data Type

Single

See Also

AlignmentPicture

Example

This sample code displays a picture 1000 twips from the right side of the caption:

```
SSFormFX1.CaptionHeight = 2000  'Make the caption big enough
SSFormFX1.CaptionPicture = LoadPicture("GOGGLES.ICO")
SSFormFX1.AlignmentPicture = 15  'Fixed from right
SSFormFX1.CaptionPictureX = 1000  '1000 twips from the right
```

Client3D Property

Applies To

SSFormFX

Description

Determines whether the client area of the form will have a 3D bevel around its edge.

Usage

[*form.*]*control.***Client3D**[= {**True**|**False**}]

Remarks

The property settings are:

Setting	Description
True	The client area will be beveled.
False	(Default) The client area will not have a bevel.

Note If you use this property to draw a 3D bevel around the client area of a form, be sure to set the **BackColor** property of the form to the same color as the **BevelColorFace** property to complete the 3D look.

Data Type

Integer (Boolean)

See Also

ClientBevelWidth, **ClientBevelInner**, **ClientBevelOuter**, **ClientBorderWidth**

ClientBevelInner Property

Applies To

SSFormFX

Description

Specifies the type of inside bevel for the client area of the form.

Usage

[*form.*]*control.***ClientBevelInner**[= s*etting*]

Remarks

This property is used to determine the type of inside beveling to draw around the client area of the form. This property is ignored if the **Client3D** property is **False.**

The settings for this property are:

Setting	Description
0	(Default) None. No inner bevel is drawn.
1	Inset. The inner bevel appears as if it is inset into the screen.
2	Raised. The inner bevel appears as if it is raised from the screen.

Data Type

Integer (Enumerated)

See Also

Client3D, **BevelColorFace**, **BevelColorHighlight**, **BevelColorShadow**, **ClientBevelWidth**

ClientBevelOuter Property

Applies To

SSFormFX

Description

Specifies the type of outside bevel for the client area of the form.

Usage

[*form.*]*control.***ClientBevelOuter**[= s*etting*]

Remarks

This property is used to determine the type of outside beveling to draw in the client area of the form. This property is ignored if the **Client3D** property is **False.**

The property settings are:

Setting	Description
0	None. No outer bevel is drawn.
1	Inset. The outer bevel appears as if it is inset into the screen.
2	(Default) Raised. The outer bevel appears as if it is raised from the screen.

Data Type

Integer (Enumerated)

See Also

Client3D, **BevelColorFace**, **BevelColorHighlight**, **BevelColorShadow**, **ClientBevelWidth**

ClientBevelWidth Property

Applies To

SSFormFX

Description

Sets or returns the width of the bevel (outer or inner) of the client area of the form, which determines the amount of the 3-D effect.

Usage

[*form.*]*control.***ClientBevelWidth**[= *width*]

Remarks

This property determines the number of pixels used to draw the inner and outer bevels which surround the client area of the form. The valid range for this property is 0 to 30. This property is ignored if the **Client3D** property is **False**.

Data Type

Integer

See Also

Client3D, **ClientBevelInner**, **ClientBevelOuter**

ClientBorderWidth Property

Applies To

SSFormFX

Description

Sets or returns the width of the space between the outer and inner bevels in the client area of the form.

Usage

[*form.*]*control.***ClientBorderWidth**[= *width*]

Remarks

This property determines the width in pixels between the inner and outer bevels drawn around the client area of the form. The valid range for this property is 0 to 30. This property is ignored if the **Client3D** property is **False**.

Data Type

Integer

See Also

Client3D, **ClientBevelInner**, **ClientBevelOuter**

ClientLeft, ClientTop Properties

Applies To

SSIndexTab

Description

Returns the X and/or Y coordinate of the left and/or top of the client area of the Index Tab control.

Usage

[*form.*]*control.***ClientLeft**

[*form.*]*control.***ClientTop**

Remarks

The client area of an Index Tab control is the area on which you can place other controls for each tab. The value returned by these properties is in the scale mode of the Index Tab controls container.

These properties can be useful to size controls on a tab based on the size of the client are of the Index Tab control.

Data Type

Integer (Boolean)

See Also

ClientWidth, **ClientHeight**

ClientWidth, ClientHeight Properties

Applies To

SSIndexTab

Description

Returns the width and/or height of the client area of the Index Tab control.

Usage

[*form.*]*control.***ClientWidth**

[*form.*]*control.***ClientHeight**

Remarks

The client area of an Index Tab control is the area on which you can place other controls for each tab. The value returned by these properties is in the scale mode of the Index Tab controls container.

These properties can be useful to size controls on a tab based on the size of the client are of the Index Tab control.

Data Type

Integer (Boolean)

See Also

ClientLeft, **ClientTop**

ClientMove Property

Applies To

SSFormFX

Description

Determines whether the form can be moved by clicking on the client area.

Usage

[*form.*]*control.***ClientMove**[= {**True**|**False**}]

Remarks

The property settings are:

Setting	Description
True	The form can be moved by clicking on the client area and dragging the mouse.
False	(Default) The form can only be moved only by clicking in the caption area.

Data Type

Integer (Boolean)

See Also

LockMove

ControlBox3D Property

Applies To

SSFormFX

Description

Determines whether the control box on the form will be 3D.

Usage

[*form.*]*control.***ControlBox3D**[= {**True**|**False**}]

Remarks

The property settings are:

Setting	Description
True	The control box will be 3D.
False	(Default) The control box will not be 3D.

The **StdCaption** property must be set to **False** for this property to take effect.

Data Type

Integer (Boolean)

See Also

Caption3D, **CaptionButtonsPic**

ControlBoxWidth Property

Applies To

SSFormFX

Description

Sets the width of the control box.

Usage

[*form.*]*control*.**ControlBoxWidth**[= *width*]

Remarks

This property sets the width in pixels of the control box in the caption area of the form. By setting this property, the width of the control box can be different than that of a normal control box. If this property is set to 0, then the control will use the standard control box width.

The **StdCaption** property must be set to **False** for this property to take effect.

Data Type

Integer

See Also

ControlBox3D, **CaptionButtonsPic**, **StdButtonHeight**

Example

The following sample shows how to get a caption to appear like the standard Visual Basic toolbox window:

```
SSFormFX1.FontName = "Small Fonts"
SSFormFX1.FontSize = 6
SSFormFX1.CaptionHeight = 0 'sizes caption based on font size
SSFormFX1.ControlBoxWidth = 10 '10 pixels
```

DockRank, DockRankSeq Properties

Applies To

SSToolbar

Description

Sets or returns the position of the toolbar relative to other toolbars when initially positioned in the same docking area.

Usage

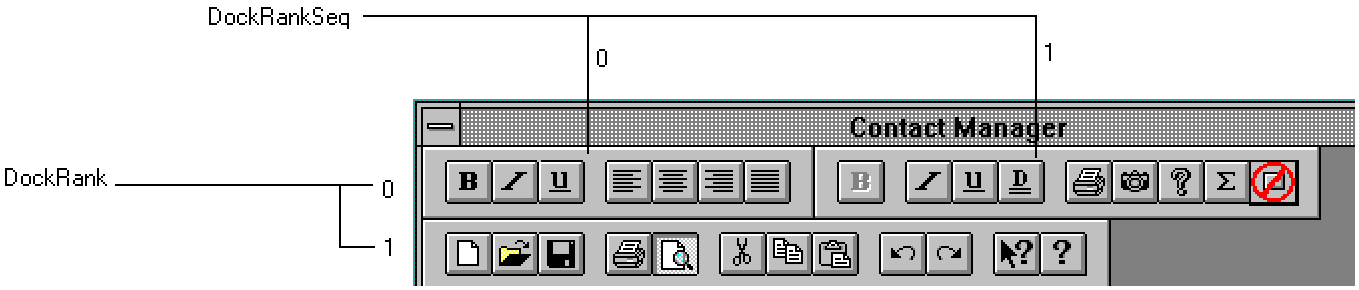
[*form.*]*control.***DockRank**[= *rank*]

[*form.*]*control.***DockRankSeq**[= *rankseq*]

Remarks

Use these properties to initially position multiple toolbars in a docking area.

When a toolbar is initially docked, the control will search for other toolbars that will also be docked in the same area. The control will then compare the **DockRank** property to that of the other toolbars and place it before or after the others. If two or more toolbars have the same setting for the **DockRank** property, the toolbar will use the **DockRankSeq** property to determine the order with the same rank.



These properties are very useful when saving the docked positions of the toolbars in an application after the user has exited. The next time the application is run, you can set these properties to their previous settings to restore the toolbar to its previous docked position. Refer to the **FloatingLeft**, **FloatingTop** and **FloatingWidthInBtns** properties to save and restore floating toolbars between program executions.

Data Type

Integer

See Also

DockStatus, **BackColorDock**

DockStatus Property

Applies To

SSToolbar

Description

Determines the docked status of the toolbar.

Usage

[*form.*]*control.***DockStatus[** = s*etting*]

Remarks

This property can be set programmatically to dock a toolbar or cause it to float.

If you set this property to 'None (Floating)', the toolbar will float and be positioned to the coordinates where the toolbar was last floating during the current execution of the application. To specify where a toolbar will initially float, use the **FloatingLeft** and **FloatingTop** properties.

Setting this property to any other setting will cause the toolbar to dock in one of the docking areas. The toolbar will then be positioned based on the **DockRank** and **DockRankSeq** properties.

You can set this property at design time to specify where the toolbar will be positioned initially at runtime.

The settings for this property are:

Setting	Description
0	(Default) Top - the toolbar will be docked along the top.
1	Left - the toolbar will be docked along the left side.
2	Bottom - the toolbar will be docked along the bottom.
3	Right - the toolbar will be docked along the right side.
4	None (Floating) - the toolbar will not be docked.

Data Type

Integer (Enumerated)

See Also

DockRank, **DockRankSeq**, **FloatingLeft**, **FloatingTop**

FloatingCaption Property

Applies To

SSToolbar

Description

Specifies the caption of a toolbar when it is floating.

Usage

[*form.*]*control.***FloatingCaption**[= *text*]

Remarks

This property sets the caption text for the toolbar when it is floating.

Data Type

String

See Also

FloatingCaptionType

FloatingCaptionType Property

Applies To

SSToolbar

Description

Determines the look of the floating toolbar's caption area.

Usage

[*form.*]*control.***FloatingCaptionType**[= s*etting*]

Remarks

The settings for this property are:

Setting	Description
0	None. No caption will be displayed when the toolbar is floating. The user can still move the palette by clicking on any area not occupied by a button.
1	(Default) Small Caption. A small caption is used. The font name and size for the text on this type of caption is selected by the control.
2	Normal Caption. A normal caption is used.
3	Use Font. The caption will be sized based on the size of the font provided in the **FontName** property.

Data Type

Integer (Enumerated)

See Also

FloatingCaption, **FloatingControlBox**

FloatingControlBox Property

Applies To

SSToolbar

Description

Determines whether a control box will be displayed on a floating toolbar, allowing the user to close it.

Usage

[*form.*]*control.***FloatingControlBox**[= {**True**|**False**}]

Remarks

The property settings are:

Setting	Description
True	(Default) A control box is displayed in the caption area of the floating toolbar.
False	User cannot close the floating toolbar.

Note When a floating toolbar is closed via a single click on the control box, the **Visible** property is set to **False**. A toolbar can only be re-opened programmatically by setting the **Visible** property back to **True**.

Data Type

Integer (Boolean)

See Also

FloatingCaptionType

FloatingLeft, FloatingTop Properties

Applies To

SSToolbar

Description

Determines the X-coordinate/Y-coordinate of the floating toolbar relative to the left/top of the MDI client area.

Usage

[*form.*]*control.***FloatingLeft** [= *x-coordinate*

[form.]control.**FloatingTop** [= y-coordinate]

Remarks

These properties will set the X-coordinate and/or Y-coordinate for the toolbar when floating.   Changing these properties at runtime while the toolbar is floating will cause the toolbar to move to that location. You can also use these properties to initially set the coordinates of the floating toolbar.

The units specified are based on the scale mode of the MDI form on which the toolbar is placed.

These properties are very useful when saving the floating positions of the toolbars in an application after the user has exited.   The next time the application is run, you can set these properties to their previous settings to restore the toolbar to its previous floating position.

Data Type

Single

See Also

FloatingWidthInBtns, **DockStatus**

FloatingSizable Property

Applies To

SSToolbar

Description

Determines whether the size of the floating toolbar can be changed by the user.

Usage

[*form.*]*control.***FloatingSizable**[= {**True**|**False**}]

Remarks

The property settings are:

Setting	Description
True	(Default) The floating toolbar can be resized. The control will display the floating toolbar with sizable borders.
False	The floating toolbar cannot be resized. The control will display a single line border around the floating toolbar.

Data Type

Integer (Boolean)

See Also

FloatingWidthInBtns

FloatingWidth, FloatingHeight Properties

Applies To

SSToolbar

Description

Returns the width/height of the floating toolbar relative to the left/top of the MDI client area.

Usage

[*form.*]*control.***FloatingWidth**

[form.]control.**FloatingHeight**

Remarks

These properties return the width and height repectively of the toolbar when it is floating.   If the toolbar is docked, these properties will return the width and height of the toolbar according to its last floating size.

The units specified are based on the scale mode of the MDI form on which the toolbar is placed.

Data Type

Single

See Also

FloatingWidthInBtns

FloatingWidthInBtns Property

Applies To

SSToolbar

Description

Determines the width of the toolbar in number of buttons when the toolbar is floating.

Usage

[*form.*]*control.***FloatingWidthInBtns**[= *buttons*]

Remarks

The floating toolbar will automatically size itself to the number of buttons specified in this property.

This property is very useful when saving the floating size of the toolbars in an application after the user has exited. The next time the application is run, you can set this property to its previous setting to restore the toolbar to its previous floating size.

Data Type

Integer

See Also

FloatingLeft, **FloatingTop**, **FloatingSizable**

Font3D Property

Applies To

SSFormFX, SSIndexTab

Description

Sets or returns the 3D style of the caption text.

Usage

[*form.*]*control.***Font3D**[= *setting*]

Remarks

The settings for this property are:

Setting	Description
0	(Default) None. Caption is displayed flat (not 3 dimensional).
1	Raised w/light shading. Caption appears as if it is raised slightly off the screen.
2	Raised w/heavy shading. Caption appears even more raised.
3	Inset w/light shading. Caption appears as if it is inset slightly into the screen.
4	Inset w/heavy shading. Caption appears even more inset.

The Font3D property works in conjunction with all the other Font Properties. Settings 2 and 4 (heavy shading) look best with larger, bolder fonts. Dramatic effects can be created by combining the different Font3D setting with the other font properties.

Data Type

Integer (Enumerated)

See Also

Caption, TabCaption()

ForeColorActive, ForeColorInactive Properties

Applies To

SSFormFX

Description

Sets the foreground color of the caption area of the form when it is the active or inactive window.

Usage

[*form.*]*control.***ForeColorActive**[= *color*]

[*form.*]*control.***ForeColorInactive**[= *color*]

Remarks

These properties specify the color for the text in the caption area when the form is active or inactive. These properties default to the colors set in the **Windows Control Panel**.

The **StdCaption** property must be set to **False** for these properties to take effect.

Data Type

Long

See Also

BackColorActive, **BackColorInactive**

HwndToolbar Property

Applies To

SSToolbar

Description

Specifies the window handle of the toolbar.

Usage

[*form.*]*control.***HwndToolbar**[= *hwnd*]

Remarks

This property represents the window handle of the visible toolbar. This property may be different than the standard **Hwnd** property of the control, so use this when you need the handle of the actual toolbar displayed on the screen.

Note The value of this property can change at any time during execution.

This property is only available at runtime and is read-only.

Data Type

Integer

InstantClose Property

Applies To

SSFormFX

Description

Determines whether a single click on the control box will close the form.

Usage

[*form.*]*control.***InstantClose**[= {**True**|**False**}]

Remarks

The property settings are:

Setting	Description
True	The control box will close the form with a single click. The form's system menu is disabled.
False	(Default) The control box will function as normal.

The **StdCaption** property must be set to **False** for this property to take effect.

Data Type

Integer (Boolean)

See Also

ControlBoxWidth, **CaptionButtonsPic**

LockCtlKeyOverride Property

Applies To

SSFormFX

Description

Allows the user to override the **LockMove** and/or **LockSize** properties at runtime by holding down the CTRL key.

Usage

[*form.*]*control.* **LockCtlKeyOverride**[= {**True**|**False**}]

Remarks

The property settings are:

Setting	Description
True	(Default) The form can be resized or moved when the CTRL key is held down even if **LockMove** and/or **LockSize** are set to **True**.
False	**LockMove** and/or **LockSize** property cannot be overridden.

Data Type

Integer (Boolean)

See Also

LockMove, **LockSize**

LockMove Property

Applies To

SSFormFX

Description

Determines whether the form can be moved by the user at runtime.

Usage

[*form.*]*control.***LockMove**[= {**True**|**False**}]

Remarks

The property settings are:

Setting	Description
True	The form cannot be moved by the user at runtime.
False	(Default) The form can be moved.

Note: By default, the user can override this lock by holding down the CTRL key. If you do not want the user to be allowed to do this, set the **LockCtlKeyOverride** property to **False**.

Data Type

Integer (Boolean)

See Also

LockCtlKeyOverride, **LockSize**

LockSize Property

Applies To

SSFormFX

Description

Determines whether the size of the form can be changed by the user at runtime.

Usage

[*form.*]*control.***LockSize**[= {**True**|**False**}]

Remarks

The property settings are:

Setting	Description
True	The form cannot be resized.
False	(Default) The form can be resized.

Note: By default, the user can override this lock by holding down the CTRL key. If you do not want the user to be allowed to do this, set the **LockCtlKeyOverride** property to **False**.

Data Type

Integer (Boolean)

See Also

LockCtlKeyOverride, **LockMove**

MaximizeHeight, MaximizeWidth Properties

Applies To

SSFormFX

Description

Specifies the height and/or width of the form when the form is maximized.

Usage

[*form.*]*control.***MaximizeHeight**[= *Height*]

[*form.*]*control.***MaximizeWidth**[= *Width*]

Remarks

These properties specify the height and/or width of the form when maximized. Set these properties to zero for the Windows default size. The value is specified based on the scale mode of the form.

Data Type

Single

See Also

MaximizeLeft, **MaximizeTop**

Example

This sample code sets the size of the form to 3000 X 5000 twips and positioned in the center of the screen when the form is maximized:

```
SSFormFX1.MaximizeWidth  = 3000
SSFormFX1.MaximizeHeight = 5000

SSFormFX1.MaximizeTop    = (Screen.Width / 2) - (SSFormFX1.MaximizeHeight / 2)
SSFormFX1.MaximizeLeft   = (Screen.Width / 2) - (SSFormFX1.MaximizeWidth / 2)
```

MaximizeLeft, MaximizeTop Properties

Applies To

SSFormFX

Description

Specifies the X-coordinate and/or Y-coordinate of the form when the form is maximized.

Usage

[*form.*]*control.***MaximizeLeft**[= *x-coordinate*]

[*form.*]*control.***MaximizeTop**[= *y-coordinate*]

Remarks

These properties specify the X-coordinate and/or Y-coordinate of the form when the form is maximized. Set these properties to zero for the Windows default position.   The value is specified based on the scale mode of the form.

Data Type

Single

See Also

MaximizeHeight, **MaximizeWidth**

Example

This sample code sets the size of the form to 3000 X 5000 twips and positioned in the center of the screen when the form is maximized:

```
SSFormFX1.MaximizeWidth  = 3000
SSFormFX1.MaximizeHeight = 5000

SSFormFX1.MaximizeTop    = (Screen.Width / 2) - (SSFormFX1.MaximizeHeight / 2)
SSFormFX1.MaximizeLeft   = (Screen.Width / 2) - (SSFormFX1.MaximizeWidth / 2)
```

Outline Property

Applies To

SSToolbar

Description

Sets or returns whether the toolbar is displayed with a 1-pixel black border around its outer edge.

Usage

[*form.*]*control.***Outline**[= {**True**|**False**}]

Remarks

This property does not apply to a floating toolbar.

The property settings are:

Setting	Description
True	A 1-pixel black border will be drawn around the toolbar.
False	(Default) No border is drawn.

Data Type

Integer (Boolean)

Picture Property

Applies To

SSIndexTab

Description

Specifies the picture for the current tab at design time.

Usage

[*form.*]*control.***Picture**[= *picture*]

Remarks

This property sets the **TabPicture()** property of the current tab specified in the **Tab** property at design time.   Although the **TabPicture()** property can support any type of picture, the **Picture** property only supports bitmaps.

This property is available only at design time.

Data Type

String

See Also

TabPicture(), Tab

Redraw Property

Applies To

SSFormFX, SSIndexTab

Description

Temporarily defers repainting of the control.

Usage

[*form.*]*control.***Redraw**[= {**True**|**False**}]

Remarks

This property is used to defer painting when changing multiple properties so that flickering does not occur.

The property settings are:

Setting	Description
True	(Default) Will redraw the control when changes are made to the control's properties.
False	Will not redraw the control when changes are made. You must set this back to **True** so the control can redraw itself.

Data Type

Integer (Boolean)

Rows Property

Applies To

SSIndexTab

Description

Returns the total number of rows of index tabs.

Usage

[*form.*]*control.***Rows**

Remarks

This property is only available at runtime and is read-only.

Data Type

Integer

See Also

Tabs, **TabsPerRow**

ShowFocusRect Property

Applies To

SSIndexTab

Description

Determines if the focus rectangle will be displayed on a tab when the control gets focus.

Usage

[*form.*]*control.***ShowFocusRect**[= {**True**|**False**}]

Remarks

The property settings are:

Setting	Description
True	(Default) Shows the focus rectangle when a tab has focus.
False	Does not show the focus rectangle when a tab has focus.

Data Type

Integer (Boolean)

SizeMaxHeight, SizeMaxWidth Properties

Applies To

SSFormFX

Description

Specifies the maximum height/width the form can be sized to at runtime.

Usage

[*form.*]*control.***SizeMaxHeight**[= *Height*]

[*form.*]*control.***SizeMaxWidth**[= *Width*]

Remarks

These properties determine the maximum height/width that the form can be resized to at runtime. Set this to zero for no size restrictions. The value specified is based on the scale mode of the form.

Data Type

Single

See Also

LockSize, **SizeMinHeight**, **SizeMinWidth**

SizeMinHeight, SizeMinWidth Properties

Applies To

SSFormFX

Description

Specifies the minimum height/width the form can be sized to.

Usage

[*form.*]*control.***SizeMinHeight**[= *Height*]

[*form.*]*control.***SizeMinWidth**[= *Width*]

Remarks

These properties will set the minimum height/width the form can be resized to. Set this to zero for no size restrictions. The value specified is based on the scale mode of the form.

Data Type

Single

See Also

LockSize, SizeMaxHeight, SizeMaxWidth

StdButtonHeight Property

Applies To

SSFormFX

Description

Determines whether the buttons in the caption area will be the standard height.

Usage

[*form.*]*control.***StdButtonHeight**[= {**True**|**False**}]

Remarks

The property settings are:

Setting	Description
True	The buttons in the caption area will be the standard height. This includes the Maximize, Minimized, and Control box buttons. This property will not make the buttons greater than the **CaptionHeight**. However, if the **CaptionHeight** is less than the standard button height, the control will shrink the buttons to fit.
False	(Default) The buttons in the caption area will be sized using the **CaptionHeight**.

The **StdCaption** property must be set to **False** for this property to take effect.

Data Type

Integer (Boolean)

See Also

CaptionHeight

StdCaption Property

Applies To

SSFormFX

Description

Determines if the control should display a standard caption, ignoring all custom caption related properties.

Usage

[*form.*]*control.***StdCaption**[= {**True**|**False**}]

Remarks

The property settings are:

Setting	Description
True	Standard Windows caption area will be used.
False	(Default) The control will alter the caption area based on caption related property settings.

If this setting is **True**, the following properties are ignored:

- **AlignmentCaption**
- AlignmentPicture
- BackColorActive
- BackColorInactive
- Caption
- Caption3D
- CaptionButtonsPic
- CaptionHeight
- CaptionMultiLine
- CaptionPicture
- ControlBox3D
- ControlBoxWidth
- Font3D
- ForeColorActive
- ForeColorInactive
- InstantClose
- StdButtonHeight
- StdCaptionHeight

Data Type

Integer (Boolean)

StdCaptionHeight Property

Applies To

SSFormFX

Description

Determines if the FormFX control should use the Windows standard height for the caption area of the form.

Usage

[*form.*]*control.***StdCaptionHeight**[= {**True**|**False**}]

Remarks

The property settings are:

Setting	Description
True	The caption height will be the Windows standard height. The **CaptionHeight** property is ignored when this property is set to **True**.
False	(Default) The control will base the caption height on the setting of the **CaptionHeight** property.

Data Type

Integer (Boolean)

Tab Property

Applies To

SSIndexTab

Description

Returns or sets the current tab.

Usage

[*form.*]*control.***Tab**[= *tabnumber*]

Remarks

This property returns the active tab in the control. If you set this property, the control will change to the tab specified by *tabnumber*. This value is zero based.

Data Type

Integer

See Also

Caption, Picture, Tabs, TabPos, TabStart

TabBackColor(), TabForeColor() Properties

Applies To

SSIndexTab

Description

Sets the background and/or foreground color of each tab.

Usage

[*form.*]*control.***TabBackColor**(*tab*)[= *color*]

[*form.*]*control.***TabForeColor**(*tab*)[= *color*]

Remarks

This property will set the color for background and/or foreground on each tab. If the **Tabs3D** property is set to **True**, the **TabBackColor()** property is ignored and the **BevelColorFace** property is used.   To set the background color of the area not occupied by any tabs, use the standard **BackColor** property.

Data Type

Long

See Also

ActiveTabBackColor, **ActiveTabForeColor**

TabCaption() Property

Applies To

SSIndexTab

Description

Specifies the caption for each index tab.

Usage

[*form.*]*control.***TabCaption**(*tab*)[= *text*]

Remarks

This property will set the caption for the specified tab.

This property is only available at runtime. To set this property at design time, select the tab by setting the **Tab** property or by clicking on the tab with the left mouse button then set the **Caption** property. This property will be saved when the form is saved.

Data Type

String

See Also

Caption, **AlignmentCaption**, **Picture**, **AlignmentPicture**

TabCutSize Property

Applies To

SSIndexTab

Description

Sets or returns the size of the cut corners of each index tab.

Usage

[*form.*]*control.***TabCutSize**[= *size*]

Remarks

The **TabCutSize** property is set in pixels.   It determines the shape of the corner of the tabs.   The valid range for this setting is 0 - 10 pixels.

Data Type

Integer

TabEnabled() Property

Applies To

SSIndexTab

Description

Determines whether the index tab will be enabled or disabled.

Usage

[*form.*]*control.***TabEnabled**(*tab*)[= {**True**|**False**}]

Remarks

When a tab is disabled, the text on the index tab gets grayed out and the user can no longer select that tab.

The property settings are:

Setting	Description
True	(Default) Tab will be enabled.
False	Tab will be disabled.

Data Type

Integer (Boolean)

See Also

TabVisible()

TabHeight Property

Applies To

SSIndexTab

Description

Sets or returns the height of each index tab.

Usage

[*form.*]*control.***TabHeight**[= *height*]

Remarks

This property represents the height of the index tabs based on the scale mode of its container.

Data Type

Single

See Also

TabMaxWidth

TabHwnd() Property

Applies To

SSIndexTab

Description

Specifies a window handle for each index tab.

Usage

[*form.*]*control.***TabHwnd**(*tab*)[= *hwnd*]

Remarks

This property represents the Windows handle of each index tab at runtime. You can use this property to move controls between tabs using the **SetParent** Windows API call.

This property is only available at runtime and is read only.

Data Type

Integer

Example

This sample code shows how to use the same command buttons on each tab using the Windows API function SetParent:

[declarations section]

```
Declare Function SetParent Lib "USER.EXE" (ByVal hWndChild As Integer, ByVal
hWndParent As Integer) As Integer

Sub SSIndexTab_Click (PreviousTab As Integer)

   Dim NewTab As Integer
   Dim rc As Integer

   NewTab = SSIndexTab.Tab
   rc = SetParent(cmdOK.Hwnd, SSIndexTab.TabHwnd(NewTab))
   rc = SetParent(cmdCancel.Hwnd, SSIndexTab.TabHwnd(NewTab))

End Sub
```

TabMaxWidth Property

Applies To

SSIndexTab

Description

Sets or returns the maximum width of each index tab.

Usage

[*form.*]*control.***TabMaxWidth**[= *width*]

Remarks

This property represents the maximum width of each tab in the scale mode of its container. If this property is set to zero, the control will automatically size the tabs to evenly fit across the control.

Data Type

Single

See Also

Tabs, **TabsPerRow**, **TabHeight**

TabOrientation Property

Applies To

SSIndexTab

Description

Sets or returns where the tabs will appear on the control.

Usage

[*form.*]*control.***TabOrientation**[= s*etting*]

Remarks

The settings for this property are:

Setting	Description
0	(Default) Tabs on Top - Tabs will be at the top of the control.
1	Tabs on Bottom - Tabs will be at the bottom of the control.
2	Tabs on Left - Tabs will be at the left of the control
3	Tabs on Right - Tabs will be at the right of the control.

Data Type

Integer (Enumerated)

TabPicture() Property

Applies To

SSIndexTab

Description

Specifies the picture to display on each tab.

Usage

[*form.*]*control.***TabPicture**(*tab*)[= *picture*]

Remarks

This property will set the picture for the specified tab.   Valid pictures are bitmaps, icons and Windows metafiles. If you specify a Windows metafile, you must also set the height and width to display the picture via the **TabPictureMetaHeight** and **TabPictureMetaWidth** properties.

This property is only available at runtime.   To set this property at design time, select the tab by setting the **Tab** property or by clicking on the tab with the left mouse button then set the **Picture** property.   This property will be saved when the form is saved.

Data Type

Integer (Picture)

See Also

Picture, **AlignmentPicture**

TabPictureMetaHeight, TabPictureMetaWidth Properties

Applies To

SSIndexTab

Description

Sets the height and/or width of a metafile selected in the **TabPicture()** property.

Usage

[*form.*]*control.***TabPictureMetaHeight**(*tab*) [= *height*]

[*form.*]*control.***TabPictureMetaWidth**(*tab*) [= *width*]

Remarks

These properties represent the height and/or width of the metafile specified in the **TabPicture** property.

The units specified are based on the scale mode of the form.

Data Type

Single

See Also

TabPicture

TabRowOffset Property

Applies To

SSIndexTab

Description

Sets or returns the offset in pixels that each row of tabs will be indented.

Usage

[*form.*]*control.***TabRowOffset**[= *size*]

Remarks

If the control is displaying two or more rows of tabs, this property will determine the offset from one row to the next. The first row of tabs is always displayed even with the left of the front index card. The next row will start at this offset, and all remaining rows will continue to be indented increasingly by this value.

Data Type

Integer

See Also

Rows, **Tabs**, **TabsPerRow**

Tabs Property

Applies To

SSIndexTab

Description

Sets or returns the total number of tabs for the control.

Usage

[*form.*]*control.***Tabs**[= *tabnumber*]

Remarks

This property determines how many tabs will be in the control.   Use this property in conjunction with the **TabsPerRow** property to determine the number of rows to display.

Data Type

Integer

See Also

Rows, **TabsPerRow**

Example

The following code sets up an Index Tab control to have 12 tabs occupying 3 rows of tabs:

```
SSIndexTab1.Tabs = 12
SSIndexTab1.TabsPerRow = 4
```

Tabs3D Property

Applies To

SSIndexTab

Description

Determines if the Index Tab control will have a 3D look.

Usage

[*form.*]*control.***Tabs3D**[= {**True**|**False**}]

Remarks

The property settings are:

Setting	Description
True	(Default) The control will be displayed in 3D.
False	The control will not be displayed in 3D.

If this property is set to **True**, the **TabBackColor** and **ActiveTabBackColor** properties are ignored and **BevelColorFace** is used.

Data Type

Integer (Boolean)

TabSelectType Property

Applies To

SSIndexTab

Description

Determines how tabs are rearranged when a new tab is selected.

Usage

[*form.*]*control.***TabSelectType**[= s*etting*]

Remarks

Regardless of the setting below, when a tab becomes active, all associated controls placed on that tab will still appear.

The settings for this property are:

Setting	Description
0	Keep Tabs in Place - The tab remains in its place when it becomes the active tab.
1	(Default) Move Selected Tab To Front Row - This setting moves the selected tab and all other tabs on its row to the front row, cycling all tab rows so that their relative order remains the same.

Data Type

Integer (Enumerated)

TabsPerRow Property

Applies To

SSIndexTab

Description

Sets or returns the number of tabs for each row.

Usage

[*form.*]*control.***TabsPerRow**[= *tabnumber*]

Remarks

This property determines how many tabs will be on each row.   Use this property in conjunction with the **Tabs** property to determine the number of rows the control will display.

Data Type

Integer

See Also

Rows, **Tabs**

Example

The following code sets up an Index Tab control to have 12 tabs occupying 3 rows of tabs:

```
SSIndexTab1.Tabs = 12
SSIndexTab1.TabsPerRow = 4
```

TabStart Property

Applies To

SSIndexTab

Description

Determines which tab will be the active index tab when the control loads.

Usage

[*form.*]*control.***TabStart**[= *tabnumber*]

Remarks

This property determines the initial active tab when the control loads. If set to -1, then the tab that is selected when the form is saved at designtime will be the initial active tab. If set to any other number, then the tab with that number will be the initial active tab. Tab numbers are zero based and can range from 0 to (**Tabs** - 1).

This property is only available at designtime.

Data Type

Integer

Example

This sample code ensures that the first active tab will be the last one:

```
SSIndexTab1.TabStart = SSIndexTab1.Tabs - 1   'TabStart is
                                               'zero based
```

TabVisible() Property

Applies To

SSIndexTab

Description

Determines whether the tab will be visible.

Usage

[*form.*]*control.***TabVisible**(*tab*)[= {**True**|**False**}]

Remarks

The property settings are:

Setting	Description
True	(Default) Tab will be visible.
False	Tab will not be visible.

Data Type

Integer (Boolean)

See Also

TabEnabled()

Toolbar3D Property

Applies To

SSToolbar

Description

Determines whether the toolbar will be 3D.

Usage

[*form.*]*control.***Toolbar3D**[= {**True**|**False**}]

Remarks

The property settings are:

Setting	Description
True	The toolbar will be displayed in 3D.
False	(Default) The toolbar will not be 3D.

Data Type

Integer (Boolean)

See Also

BevelColorFace, **BevelColorHighlight**, **BevelColorShadow**, **BevelInner**, **BevelOuter**, **BevelWidth**, **BorderWidth**

ToolsetName Property

Applies To

SSToolbar

Description

Contains a descriptive name for the tool.

Usage

[*form.*]*control.***ToolsetName**[= *NameString*]

Remarks

This property might be useful when presenting the user with a choice of toolsets to enable customization at runtime.

This property is only available at runtime (it is indirectly settable at designtime via the Toolbar Designer).

Data Type

String

ToolsetNumBtnStates Property

Applies To

SSToolbar

Description

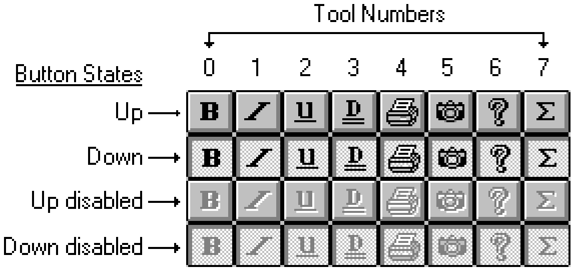
Sets or returns the number of button states included in the **ToolsetPicture** property.

Usage

[*form.*]*control.***ToolsetNumBtnStates**[= *numstates*]

Remarks

This property determines the number of states of buttons that are included in the **ToolsetPicture** property. A value of two means that only the 'up' and 'down' states are included. Three means that the 'up disabled' state is also included while a value of four means that all four states are included (as in the sample bitmap below).



This property is only available at runtime (it is indirectly settable at designtime via the Toolbar Designer).

Data Type

Integer

See Also

ToolsetPicture, **ToolsetNumTools**

ToolsetNumTools Property

Applies To

SSToolbar

Description

Sets or returns the number of tools in the **ToolsetPicture** property.

Usage

[*form.*]*control.***ToolsetNumTools**[= *tools*]

Remarks

This property determines the number of tools (columns of button images) defined in the **ToolsetPicture** property.

This property is only available at runtime (it is indirectly settable at designtime via the Toolbar Designer).

Data Type

Integer

See Also

ToolsetPicture, **ToolsetNumBtnStates**

Example

This sample code dynamically adds a new tool to the current toolset at runtime and assigns it to button number 1:

```
Dim NumTools As Integer

SSToolbar1.ToolsetPicture = LoadPicture("")

NumTools = SSToolbar1.ToolsetNumTools + 1

SSToolbar1.ToolsetNumTools = NumTools
SSToolbar1.ToolsetPicUp(NumTools - 1) = LoadPicture("TOOL_UP.BMP")
SSToolbar1.ToolsetPicDn(NumTools - 1) = LoadPicture("TOOL_DN.BMP")
SSToolbar1.ToolsetPicUp(NumTools - 1) = LoadPicture("TOOL_UP.BMP")

SSToolbar1.BtnToolNum(1) = NumTools - 1
```

ToolsetPicture Property

Applies To

SSToolbar

Description

Specifies the bitmap to use for the toolset.

Usage

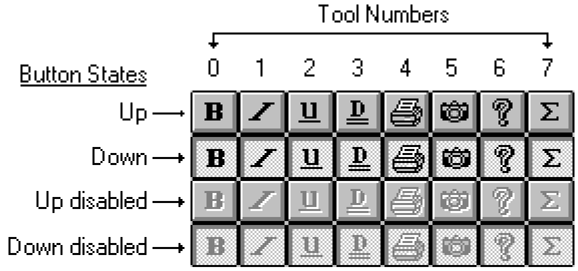
[*form.*]*control.***ToolsetPicture**[= *picture*]

Remarks

This property contains a bitmap to use as the current toolset. Toolsets are collections of related tools. Each tool represents a specific user function. The bitmap must contain an image for at least the up and down state of every tool in the toolset. The possible states are:

1. Up (required)
2. Down (required)
3. Up disabled (optional)
4. Down disabled (optional)

Note The bitmap can be created or modified with any bitmap editor including the standard PaintBrush application (PBRUSH.EXE).



When creating the bitmap you need to be aware of the following:

- The size of the image for each state of each tool in the toolset must be the same.
- You must supply at least the 'up' and 'down states' (the 'up disabled' and 'down disabled' states are optional).
- The **ToolsetNumBtnStates** property lets you specify which button states are supplied. A value of two means that only the 'up' and 'down' states are included. Three means that the 'up disabled' state is also included while a value of four means that all four states are included (as in the sample bitmap above).
- The **ToolsetNumTools** property specifies how many tools are in the set. This would be set to 8 for the above sample bitmap.
- **BtnWidth** and **BtnHeight** properties are automatically calculated based on the **ToolsetNumBtnStates** and **ToolsetNumTools** property settings and the size of the bitmap.

This property is only available at runtime.

Data Type

Integer (Picture)

See Also

ToolsetNumTools, ToolsetToolID(), **ToolsetToolHelp**(), **ToolsetToolDesc**(), **ToolsetToolType**(),**ToolsetToolGroup**(), **ToolsetToolExclusive**(), **ToolsetToolAllowAllUp**(), **ToolsetToolMnemonic**()

Example

This sample code assigns a bitmap to the **ToolsetPicture** property and configures it to have 8 tools and 2 states (up and down only), and set the type of the tools to be 'Push Button':

```
Dim I As Integer

Tb1.ToolsetNumBtnStates = 2
Tb1.ToolsetNumTools = 8
Tb1.ToolsetPicture = LoadPicture("MYTOOLS.BMP")

For i = 0 to Tb1.ToolsetNumTools - 1
       Tb1.ToolsetToolType(i) = 0   'Push Button
Next i
```

ToolsetToolAllowAllUp() Property

Applies To

SSToolbar

Description

Determines whether all buttons in a group can be in the 'up' position.

Usage

[*form.*]*control.***ToolsetToolAllowAllUp**(*tool*)[= {**True**|**False**}]

Remarks

The property settings are:

Setting	Description
True	All buttons in the current picture group may be in the 'up' position.
False	(Default) At least one button in the current picture group must be depressed.

Note When this property is set for a button in a group, then the property is automatically set to the same value for all the other buttons in the group.

If the **ToolsetToolAllowAllUp()** property is set to **False**, no check will be made by the button to insure that at least one button is depressed when the toolbar on which the button resides is loaded. It is up to you to set the initial state of the **Value** property for one of the buttons in the group to **True**.

This property is ignored if the **ToolsetToolExclusive()** property for the same group is set to **False** or the **ToolsetToolType()** property for this tool is set to 'Push'.

This property is only available at runtime. However, it can be initially set at design time using the Toolbar Designer.

Data Type

Integer (Boolean)

See Also

ToolsetToolGroup()

ToolsetToolDesc() Property

Applies To

SSToolbar

Description

Contains a description of the tool.

Usage

[*form.*]*control.***ToolsetToolDesc**(*tool)*[= *string*]

Remarks

This property is used to store a description of a tool.

This property is only available at runtime.   However, it can be initially set at design time using the Toolbar Designer.

Data Type

String

See Also

ToolsetToolHelp()

Example

This sample code shows how to display the tool's description in a status bar when the mouse moves over a button:

```
Sub SSToolbar1_MouseEnter(ToolID As String, ToolNum As Integer, Btn As Integer)

    pnlStatus.Caption = SSToolbar1.ToolsetToolDesc(ToolNum)

End Sub
```

ToolsetToolExclusive() Property

Applies To

SSToolbar

Description

Determines if only one button in the same group should be toggled down at any time.

Usage

[*form.*]*control.***ToolsetToolExclusive**(*tool*)[= {**True**|**False**}]

Remarks

The property settings are:

Setting	Description
True	Only one button in the same group can be toggled down (**BtnValue** = **True**) at any time.
False	(Default) Buttons within the same group toggle independently.

Note When this property is set for a button in a group, then the property is automatically set to the same value for all the other buttons in the group.

When a button is pressed and this property is set to **True**, the control will toggle all other 'toggle' buttons in the same logical group (based on the **ToolsetToolGroup()** property) to their 'up' positions. If this property is set to **False** then the **ToolsetToolAllowAllUp()** property is ignored.

This property is only available at runtime. However, it can be initially set at design time using the Toolbar Designer.

Data Type

Integer (Boolean)

See Also

ToolsetToolExclusive(), ToolsetToolAllowAllUp()

ToolsetToolGroup() Property

Applies To

SSToolbar

Description

Sets or returns the group the tool belongs to.

Usage

[*form.*]*control.***ToolsetToolGroup**(*tool*)[= *groupnumber*]

Remarks

The amount of space between buttons in a group is defined by the **BtnGroupSpacing** property.   Certain properties apply to buttons in groups (refer to the property definitions in this chapter for a full description):

- **ToolsetToolExclusive()**
- **ToolsetToolAllowAllUp()**
- **BtnGroupSpacing**

This property is only available at runtime.   However, it can be initially set at design time using the Toolbar Designer.

Data Type

Integer

See Also

ToolsetToolExclusive(), ToolsetToolAllowAllUp(), BtnGroupSpacing

ToolsetToolHelp() Property

Applies To

SSToolbar

Description

Sets or returns the help text for the tool (which is used to display balloon help).

Usage

[*form.*]*control.***ToolsetToolHelp**(*tool*)[= *string*]

Remarks

This property specifies what help message is associated with the specified tool. When the **BalloonHelp** property is set to **True**, the string in **ToolsetToolHelp()** property will appear when the mousepointer pauses over a button.   If this property is not defined for a tool, no balloon help will appear.

This property is only available at runtime or can be set at design time using the Toolbar Designer.

Data Type

String

See Also

BalloonHelp, **BalloonHelpDelay**, **ToolsetToolDesc()**

ToolsetToolID() Property

Applies To

SSToolbar

Description

Sets or returns a unique ID for a tool.

Usage

[*form.*]*control.***ToolsetToolID**(*tool*)[= *String*]

Remarks

The string in this property represents a unique ID used to identify a tool from code at runtime. It can be a maximum of 16 characters. This string is passed to you in the **Click**, **Help**, **MouseEnter** and **MouseExit** events.

This property is only available at runtime or can be set at design time using the Toolbar Designer.

Data Type

String

ToolsetToolMnemonic() Property

Applies To

SSToolbar

Description

Sets or returns the mnemonic character associated with a tool.

Usage

[*form.*]*control.***ToolsetToolMnemonic**(*tool*)[= *character*]

Remarks

This property will set or return the mnemonic character for the tool specified. By setting a mnemonic for a tool, the user can click a button by hitting ALT and the character specified in **ToolsetToolMnemonic()**. This will cause the **Click** event of the button to fire.   This property is generally used when a caption is included in the picture to allow the user to 'hotkey' the button.

Note   You should be aware that mnemonic characters only work within the current active form. Therefore, if a MDI child form is active, the mnemonic characters for toolbars that are on the MDI parent will not operate.

This property is only available at runtime or can be set at design time using the Toolbar Designer.

Data Type

String (Single Character Only)

ToolsetToolPicDn(), ToolsetToolPicUp() Properties

Applies To

SSToolbar

Description

Sets or returns the image of the tool for the button's up or down state.

Usage

[*form.*]*control.***ToolsetToolPicDn**(*tool*)[= *picture*]

[*form.*]*control.***ToolsetToolPicUp**(*tool*)[= *picture*]

Remarks

These properties can be used to extract or set the bitmap images used on a button in the up and/or down positions.

Note These properties are not kept as separate bitmaps by the toolbar and therefore do not consume excessive system resources. They are used just like the **GraphicCell**() property in the Picture Clip control (included in the Visual Basic Professional Edition) except that they are not read-only.

Keep in mind that when you set any of these individual picture properties you are in fact updating an area of the larger (all inclusive) **ToolsetPicture** bitmap.

This property is only available at runtime.

Data Type

Integer (Picture)

See Also

ToolsetPicture, **ToolsetToolPicDnDis()**, **ToolsetToolPicUpDis()**

Example

The following code is a routine that gets passed two arrays of bitmap filenames each for the up and down image to use to create a toolset dynamically at runtime:

```
Sub CreateToolset(BmpUp() As String, BmpDn() As String)

   Dim I As Integer

   Tb1.ToolsetPicture = LoadPicture("")

   Tb1.ToolsetNumBtnStates = 2
   Tb1.ToolsetNumTools     = UBound(BmpUp)

   For i = 0 to Tb1.ToolsetNumTools - 1
       Tb1.ToolsetToolPicUp(i%) = LoadPicture(BmpUp(i))
       Tb1.ToolsetToolPicDn(i%) = LoadPicture(BmpDn(i))
   Next I

End Sub
```

Note The first bitmap applied determines the width and height of all tools in the toolset, so all the bitmaps should be the same size or else buttons will not display the desired images.

ToolsetToolPicDnDis(), ToolsetToolPicUpDis() Properties

Applies To

SSToolbar

Description

Sets or returns the picture for the tool for its disabled-up and disabled-down states.

Usage

[*form.*]*control.***ToolsetToolPicDnDis** (*tool*)[= *picture*]

[*form.*]*control.***ToolsetToolPicUpDis** (*tool*)[= *picture*]

Remarks

These properties can be used to extract or set the bitmap images used on a button in the disabled-up and/or disabled-down positions. Since these two states are optional (based on the **ToolsetNumBtnStates** property) and may not exist in the bitmap specified in the **ToolsetPicture** property, these properties may not have a value.

Note These properties are not kept as separate bitmaps by the toolbar. They are used just like the **GraphicCell**() property in the Picture Clip control (included in the Visual Basic Professional Edition) except that they are not read-only.

Keep in mind that when you set any of these individual picture properties you are in fact updating an area of the larger (all inclusive) **ToolsetPicture** bitmap.

These properties are only available at runtime and are read-only.

Data Type

Integer (Enumerated)

See Also

ToolsetNumBtnStates, **ToolsetPicture**, **ToolsetToolPicDn()**, **ToolsetToolPicUp()**

ToolsetToolType() Property

Applies To

SSToolbar

Description

Determines the type of each tool to be either a push button or a toggle button.

Usage

[*form.*]*control.***ToolsetToolType**(*tool*)[= *setting*]

Remarks

The settings for this property are:

Setting	Description
0	(Default) Push. A button associated with this tool will depress and automatically return to the up position when clicked.
1	Toggle. A button associated with this tool will toggle between up and down states when clicked.

This property is only available at runtime or can be set at design time using the Toolbar Designer.

Data Type

Integer (Enumerated)

See Also

ToolsetToolGroup()

Events Reference

The following is a complete reference of all the custom events in the Designer Widgets custom controls. For all other standard events, see the *Microsoft Visual Basic Language Reference***.**

Related Topics:

Click Event (SSToolbar)
Click Event (SSIndexTab)
DockStatusChanged Event
Help Event
MouseEnter Event
MouseExit Event
ToolbarClosed Event
ToolbarResized Event

Click Event (SSToolbar)

Applies To

SSToolbar

Description

Occurs when the user presses a button on the toolbar or a button tool's associated mnemonic character (see **ToolsetToolMnemonic()** in chapter 11) is entered via the keyboard.

Syntax

Sub *control*_**Click** ([*Index* **As Integer**] *ToolID* As **String**, *ToolNum* As **Integer**, *Btn* As **Integer**, *Value* As **Integer**)

Remarks

The event parameters are:

Parameter	Description
Index	Uniquely identifies a control if it is in a control array.
ToolID	The ToolID of the tool that is associated with the button that was clicked. This is the ID specified in the **ToolsetToolID()** property of the tool.
ToolNum	The zero based index of the button's tool in the toolset.
Btn	The button's index # (0 is the leftmost button on toolbar).
Value	Reflects the button's state after the click. This parameter is **True** if the button is down, **False** if up.

Example

This sample shows how to identify a button based on the ToolID parameter as defined in the toolset. This sample uses the toolset TSBASIC.BMP:

```
Sub SSToolbar1_Click (ToolID As String, ToolNum As Integer,
                     Btn As Integer, Value As Integer)

       Select Case ToolID
           Case "ID_NEW"
               MsgBox "This invokes the New File function"
           Case "ID_OPEN"
               MsgBox "This invokes the Open File function"
           Case "ID_SAVE"
               MsgBox "This invokes the Save File function"
           Case "ID_PRINT"
               MsgBox "This invokes the Print function"
           Case "ID_PREVW"
               MsgBox "This invokes Print Preview"
           Case "ID_CUT"
               MsgBox "This invokes the Cut function"
           Case "ID_COPY"
               MsgBox "This invokes the Copy function"
           Case "ID_PASTE"
               MsgBox "This invokes the Paste function"
           Case "ID_UNDO"
               MsgBox "This invokes the Undo function"
           Case "ID_REDO"
```

```
               MsgBox "This invokes the Redo function"
           Case "ID_HELP1"
               MsgBox "This invokes the Item Help function"
           Case "ID_HELP2"
               MsgBox "This invokes General Help"
       End Select

End Sub
```

Click Event (SSIndexTab)

Applies To

SSIndexTab

Description

Occurs when the user selects an index tab.

Syntax

Sub *control*_**Click** ([*Index* **As Integer**] *PreviousTab* As **Integer**)

Remarks

The event parameters are:

Parameter	Description
Index	Uniquely identifies a control if it is in a control array.
PreviousTab	Identifies the tab that was previously active.

When the user clicks on a tab, that tab becomes the active tab and all controls placed on that tab at design time become visible.   The **Click** event notifies you when the user clicks on a tab to make it the active tab.

Example

This sample code shows how to use the same command buttons on each tab using the Windows API function SetParent:

[declarations section]

```
Declare Function SetParent Lib "USER.EXE" (ByVal hWndChild
       As Integer, ByVal hWndParent As Integer) As Integer

Sub SSIndexTab_Click (PreviousTab As Integer)

   Dim NewTab As Integer
   Dim rc As Integer

   NewTab = SSIndexTab.Tab
   rc = SetParent(cmdOK.Hwnd, SSIndexTab.TabHwnd(NewTab))
   rc = SetParent(cmdCancel.Hwnd, SSIndexTab.TabHwnd(NewTab))

End Sub
```

DockStatusChanged Event

Applies To

SSToolbar

Description

Occurs after the toolbar has changed its docked position.

Syntax

Sub *control*_**DockStatusChanged** ([*Index* **As Integer**] *PreviousDock* As **Integer**)

Remarks

This event will occur after the toolbar has changed docked positions and will also occur when the toolbar becomes a floating toolbar.   To get the new dock status, use the **DockStatus** property.

The event parameters are:

<table>
<tr><th>Parameter</th><th>Description</th></tr>
<tr><td><i>Index</i></td><td>Uniquely identifies a control if it is in a control array.</td></tr>
<tr><td><i>PreviousDock</i></td><td>Identifies the previous dock status of the toolbar.
<table>
<tr><th>Setting</th><th>Description</th></tr>
<tr><td>0</td><td>Top - The toolbar was previously docked in the top docking area.</td></tr>
<tr><td>1</td><td>Left - The toolbar was previously docked in the left docking area.</td></tr>
<tr><td>2</td><td>Bottom - The toolbar was previously docked in the bottom docking area.</td></tr>
<tr><td>3</td><td>Right - The toolbar was previously docked in the right docking area.</td></tr>
<tr><td>4</td><td>None (Floating) - The toolbar was previously floating.</td></tr>
</table></td></tr>
</table>

Help Event

Applies To

SSToolbar

Description

This event is triggered when the balloon help is activated.

Syntax

Sub *control*_**Help** ([*Index* **As Integer**], *ToolID* As **String**, *ToolNum* As **Integer**, *Btn* As **Integer,** *X* As **Long**, *Y* As **Long**, *RtnCancelDisplay* As **Integer**)

Remarks

This event gets triggered when the balloon help is about to be displayed. You can bypass the display by setting the *RtnCancelDisplay* parameter to **True**. This event only gets fired if the **BalloonHelp** property is set to **True**.

The balloon help text will disappear when a key is pressed or the user moves the mouse off the button.

Parameter	Description
Index	Uniquely identifies a control if it is in a control array.
ToolID	The ToolID of the tool that is associated with this button. This is the 8-character ID specified in the **ToolsetToolID()** property of the tool.
ToolNum	The zero based index of the tool in the toolset.
Btn	The button's index # (0 is the leftmost button on toolbar).
X	The X-coordinate of the window that will display the balloon help. Use this coordinate to display your own balloon help window. The coordinate is specified in pixels relative the entire screen.
Y	The Y-coordinate of the window that will display the balloon help. Use this coordinate to display your own balloon help window. The coordinate is specified in pixels relative the entire screen.
RtnCancelDisplay	This parameter is used to bypass the display of the balloon help text. If you want to bypass the display, set this parameter to **True**.

Example

This sample code shows how to display the tool's description in a status bar when the balloon help is activated. To clear the text from the status bar, respond to the **MouseExit** event:

```
Sub SSToolbar1_Help(ToolID As String, ToolNum As Integer,
                    Btn As Integer, X As Long, Y As Long,
                    RtnCancelDisplay As Integer)

    pnlStatus.Caption = SSToolbar1.ToolsetToolDesc(ToolNum)

End Sub
```

MouseEnter Event

Applies To

SSToolbar

Description

Occurs when the mouse pointer enters a button.

Syntax

Sub *control*_**MouseEnter** ([*Index* **As Integer**] *ToolID* As **String**, *ToolNum* As **Integer**, *Btn* As **Integer**)

Remarks

This event will occur when the mousepointer moves over a button on the toolbar.

The event parameters are:

Parameter	Description
Index	Uniquely identifies a control if it is in a control array.
ToolID	The ToolID of the tool that is associated with this button. This is the ID specified in the **ToolsetToolID()** property of the tool.
ToolNum	The zero based index of the tool in the toolset.
Btn	The button's index # (0 is the leftmost button on toolbar).

See Also

MouseExit

Example

This sample code shows how to display the tool's description in a status bar when the mouse is moved over a button:

```
Sub SSToolbar1_MouseEnter(ToolID As String,
                    ToolNum As Integer, Btn As Integer)

    pnlStatus.Caption = SSToolbar1.ToolsetToolDesc(ToolNum)

End Sub
```

MouseExit Event

Applies To

SSToolbar

Description

Occurs when the mouse pointer exits a button.

Syntax

Sub *control*_**MouseExit** ([*Index* **As Integer**] *ToolID* As **String**, *ToolNum* As **Integer**, *Btn* As **Integer**)

Remarks

This event will occur when the mouse pointer leaves a button on the toolbar.

The event parameters are:

Parameter	Description
Index	Uniquely identifies a control if it is in a control array.
ToolID	The ToolID of the tool that is associated with this button. This is the ID specified in the **ToolsetToolID()** property of the tool.
ToolNum	The zero based index of the tool in the toolset.
Btn	The button's index # (0 is the leftmost button on toolbar).

See Also

MouseEnter

Example

This sample code shows how to clear the tool's description in a status bar (assuming it was displayed in either the **Help** or **MouseEnter** event) when the mouse is moved off a button:

```
Sub SSToolbar1_MouseExit(ToolID As String,
                    ToolNum As Integer, Btn As Integer)

    pnlStatus.Caption = ""

End Sub
```

ToolbarClosed Event

Applies To

SSToolbar

Description

Occurs when the user closes the floating toolbar by clicking on the control box.

Syntax

Sub *control*_**ToolbarClosed** ([*Index* **As Integer**])

Remarks

The *Index* parameter uniquely identifies the control if it is in a control array.

When the user closes a floating toolbar, the **Visible** property of the toolbar is set to **False**. This event will occur after the user closes the toolbar.

Example

The following sample code maintains a list of "closed" toolbars in a global Type array variable so that later in the program, the user can select from a list of toolbars to re-open:

```
Sub SSToolbar1_ToolbarClosed(Index As Integer)

   gToolbarInfo(Index).Closed = True

End Sub
```

ToolbarResized Event

Applies To

SSToolbar

Description

Occurs when the floating toolbar has been resized at runtime by the user.

Syntax

Sub *control*_**ToolbarResized** ([*Index* **As Integer**])

Remarks

The *Index* parameter uniquely identifies the control if it is in a control array.

This event will occur after the floating toolbar has been resized.   A toolbar can only be resized at runtime if the **FloatingSizable** property is set to **True**.

Technical Specifications

All of the following information is subject to change. Please check the README.TXT file for any updates.

System Requirements

- Microsoft Windows version 3.1 or higher.
- Microsoft Visual Basic version 3.0 or higher. (See hardware and system requirements for installing Visual Basic in the Setup chapter of the Microsoft Visual Basic Programmer's Guide).
- At least 2 megabytes of available space on your hard disk.

Included Files

The following table page gives a brief description of the files that are installed on your hard disk during the Setup process (see Chapter 2).

Files installed in the **\WINDOWS\SYSTEM** directory (optional):

Filename(s)	Description
SSDOCKTB.VBX	Contains the Dockable Toolbar control.
SSIDXTAB.VBX	Contains the Index Tab control.
SSFORMFX.VBX	Contains the FormFX control.

Files installed in the **\WINDOWS** directory:

Filename(s)	Description
SSDESWDG.LIC	The Designer Widgets design time license file.
SSDESWDG.INI	The Designer Widgets INI file.

Files installed in the **\SSDESWDG** directory (or whatever directory you specified during installation):

Filename(s)	Description
README.TXT	Contains updated information not found in this manual.
SSDESWDG.BAS	The declarations for the Designer Widgets API functions and the constant declarations for various Designer Widgets settings.
SSDESWDG.HLP	The Designer Widgets online help file.
SSDOCKTB.VBX	Contains the Dockable Toolbar control.
SSIDXTAB.VBX	Contains the Index Tab control.
SSFORMFX.VBX	Contains the FormFX control.

Sample projects directories installed under the **\SSDESWDG\SAMPLES** directory:

Filename(s)	Description
CHAPTER*\*.*	Contains the chapter examples.
TOOLSETS\*.*	Contains sample toolsets.

Error Messages

This appendix shows a list of trappable errors that could occur at runtime when using the Designer Widgets custom controls. The constant declarations for these values can be found in the SSDESWDG.BAS file that comes with Designer Widgets.

Error Number	**Description**
30002	SS_ERR_BEVELWIDTH1 **Bevel Width must be from 0 to 30** You tried setting one of these properties with a value outside its valid range.
30003	SS_ERR_BORDERWIDTH **Border Width must be from 0 to 30** You tried setting one of these properties with a value outside its valid range.
30004	SS_ERR_BADEXTENT1 **The property value must be between 0 and *n*** You tried setting this property with a value outside its valid range.
30005	SS_ERR_BADEXTENT2 **The property value must be greater than or equal to 0** You tried setting this property with a value outside its valid range.
30006	SS_ERR_BADEXTENT3 **The property value must be greater than 0** You tried setting this property with a value outside its valid range.
30007	SS_ERR_NUMSTATES **ToolsetNumBtnStates property can only be set to 2, 3 or 4** The **ToolsetNumBtnStates** property can only be set to 2, 3 or 4. Try specifying one of these values.
30008	SS_ERR_NOVISIBLE **Must have at least 1 visible index tab** You tried to set the last visible tab's **TabVisible()** property to **False**. There must be at least one visible tab at all times. Try setting another tab's **TabVisible()** property to **True** then set this one to **False**.
30009	SS_ERR_DELAYVALUE **BalloonHelpDelay must be from 0 to 5000** You tried setting one of these properties with a value outside its valid range.
30010	SS_ERR_FORMAT1

Only Picture Formats '.BMP' & '.ICO' supported.

You specified an invalid picture format for the property.
Try setting this property to a picture with a valid format.

30011 SS_ERR_FORMAT2

Only Picture Format '.BMP' supported.

You specified an invalid picture format for the property.
Try setting this property to a picture with a valid format.

30012 SS_ERR_FORMAT3

Only Picture Format '.BMP' supported at design time, use the TabPicture() property at runtime instead

You can only set the **Picture** property to a bitmap at design time. To specify a picture with this format to appear on this tab, use the **TabPicture()** property at runtime instead.

30013 SS_ERR_TABSEXIST

Invalid property value - controls exist on tab *n* and must first be removed before decreasing the Tabs property to this value

This error occurs if controls exist on a tab number that is greater than the value you specified. For example, you placed controls on tab number 10 and tried to set the **Tabs** property to 7. The control will not delete the child controls for you. It is up to you to remove all controls first.