

(About) Property

Applies To

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.

(Custom) Property

Applies To

Description

Displays the property pages for the control.

Usage

Right-click on the object and choose 'Property Pages' from the pop-up menu, or click on the ellipses ('...') button next to the property text to activate the property pages dialog box.

Remarks

Available only at design time. The property pages dialog provides access to the properties of the control in design environments which do not have a property sheet. Property Pages also provide access to certain properties at design-time which are not available from the property sheet.

For more information see **Property Pages**

ActiveThread Constants

AlignFrameText Constants

Constant	Value	Description
ssLeftJustify	0	Left align text.
ssRightJustify	1	Right align text.
ssCenter	2	Center text.

AlignPanelText Constants

Constant	Value	Description
ssLeftTop	0	Text to left and top.
ssLeftMiddle	1	Text to left and middle.
ssLeftBottom	2	Text to left and bottom.
ssRightTop	3	Text to right and top.
ssRightMiddle	4	Text to right and middle.
ssRightBottom	5	Text to right and bottom.
ssCenterTop	6	Text to center and top.
ssCenterMiddle	7	Text to center and middle.
ssCenterBottom	8	Text to center and bottom.

AlignTo Constants

Constant	Value	Description
ssTextRight	0	Text to right.
ssTextLeft	1	Text to left.

AutoSize Constants (SSSplitter)

Constant	Value	Description
ssAutoSizeNone	0	No autosizing.
ssAutoSizeFillContainer	1	Autosize Splitter control to form or container.

AutoSizeButton Constants

Constant	Value	Description
ssNone	0	No autosizing.
ssPictureToButton	1	Autosize picture to button.
ssButtonToPicture	2	Autosize button to picture.

AutoSizePanel Constants

Constant	Value	Description
-----------------	--------------	--------------------

ssNone	0	No autosizing.
ssWidthToCaption	1	Autosize panel width to caption.
ssHeightToCaption	2	Autosize panel height to caption.
ssChildToPanel	3	Autosize child form to panel.

BackStyle Constants

Constant	Value	Description
ssOpaque	0	Opaque background.
ssTransparent	1	Transparent background.

Bevel Constants

Constant	Value	Description
ssNone	0	No inner or outer bevel.
ssInset	1	Inset inner or outer bevel.
ssRaised	2	Raised inner or outer bevel.

BorderStyle Constants (SSSplitter)

Constant	Value	Description
ssBorderStyleNone	0	Control will have no border
ssBorderStyleFixedSingle	1	Control will have a single pixel black border
ssBorderStyleInset	2	Control will appear inset into the background
ssBorderStyleRaised	3	Control will appear raised above the background

ButtonStyle Constants

Constant	Value	Description
ssFollowOS	0	Button follows default OS button appearance.
ssWin95	1	Windows 95 / NT 4.0 button appearance.
ssWin3X	2	Windows 3.X / NT 3.X button appearance.
ssActiveBorders	3	Button borders appear when mouse is over button.
ssBorderless	4	No button borders.

CaptionStyle Constants

Constant	Value	Description
ssStandard	0	Caption text appears on a single line.
ssWrapped	1	Caption text can wrap to multiple lines.

CheckBoxValue Constants

Constant	Value	Description
----------	-------	-------------

ssCBUnchecked	0	Check box is not checked.
ssCBChecked	1	Check box is checked.
ssCBGrayed	2	Check box is grayed (indeterminate.)

Clipboard Constants

Constant	Value	Description
ssCFText	1	Clipboard format is pure text (TXT).
ssCFBitmap	2	Clipboard format is bitmap (BMP).
ssCFMetafile	3	Clipboard format is standard metafile (WMF).
ssCFDIB	8	Clipboard format is device-independent bitmap.
ssCFPalette	9	Clipboard format is color palette.
ssCFEMetafile	14	Clipboard format is enhanced metafile (EMF).
ssCFFiles	15	Clipboard format is a list of filenames (Explorer).
ssCFRTF	-16639	Clipboard format is rich text w/formatting (RTF).

DragOver Constants

Constant	Value	Description
ssEnter	0	OLEDragEnter - mouse cursor has entered control.
ssLeave	1	OLEDragLeave - mouse cursor has left control.
ssOver	2	OLEDragOver - mouse cursor is inside control.

FloodFillStyle Constants

Constant	Value	Description
ssSegmented	0	Panel fill will be segmented.
ssSolid	1	Panel fill will be solid.

FloodType Constants

Constant	Value	Description
ssNone	0	No flood.
ssLeftToRight	1	Flood from left to right.
ssRightToLeft	2	Flood from right to left.
ssTopToBottom	3	Flood from top to bottom.
ssBottomToTop	4	Flood from bottom to top.
ssWideningCircle	5	Flood in widening circle.

Font3D Constants

Constant	Value	Description
ssNone	0	No 3-D text font.
ssRaisedLight	1	Font raised with light shading.

ssRaisedHeavy	2	Font raised with heavy shading.
ssInsetLight	3	Font inset with light shading.
ssInsetHeavy	4	Font inset with heavy shading.

MarqueeDirection Constants

Constant	Value	Description
ssMDRightToLeft	0	Text moves from right to left.
ssMDLeftToRight	1	Text moves from left to right.
ssMDTopToBottom	2	Text moves from top to bottom.
ssMDBottomToTop	3	Text moves from bottom to top.

MarqueeStyle Constants

Constant	Value	Description
ssNoneMarquee	0	Text is static.
ssScrollingMarquee	1	Text moves constantly in one direction.
ssSlidingMarquee	2	Text moves and then stops.
ssBlinkingMarquee	3	Text flashes on and off.
ssBouncingMarquee	4	Text moves and reverses direction.

OLEDrag Constants

Constant	Value	Description
ssOLEDragManual	0	OLEDrag mode is manual.
ssOLEDragAutomatic	1	OLEDrag mode is automatic.

OLEDrop Constants

Constant	Value	Description
ssOLEDropNone	0	OLEDrop mode is not supported.
ssOLEDropManual	1	OLEDrop mode is manual.

OLEDropEffect Constants

Constant	Value	Description
ssOLEDropEffectNone	0	OLEDrop mode is not supported.
ssOLEDropEffectCopy	1	OLEDrop data is being copied.
ssOLEDropEffectMove	2	OLEDrop data is being moved
ssOLEDropEffectScroll	-2147483648 (&H80000000)	OLEDrop data causes scrolling.

PictureAlignment Constants

Constant	Value	Description
ssPALeftOfCaption	0	Picture aligned to left of text.
ssPARightOfCaption	1	Picture aligned to right of text.
ssPAJustify	2	Picture aligned to control edge opposite of text.

PictureBackground Constants

Constant	Value	Description
ssCentered	0	Background picture is centered at actual size.
ssStretched	1	Background picture is stretched to fill control.
ssTiled	2	Background picture is tiled to fill control.

PictureDnChange Constants

Constant	Value	Description
ssNoChange	0	Use Up bitmap with no change.
ssDither	1	Dither Up bitmap.
ssInvert	2	Invert Up bitmap.

Shadow Color Constants

Constant	Value	Description
ssDarkGrey	0	Dark gray shadow.
ssBlack	1	Black shadow.

ShadowStyle Constants

Constant	Value	Description
ssInset	0	Shadow inset.
ssRaised	1	Shadow raised.

SplitterBarAppearance (SSSplitter)

Constant	Value	Description
ssSplitterBarBorderless	0	Splitter bars have no border.
ssSplitterBarFlat	1	Splitter bars have flat, 1-pixel borders.
ssSplitterBar3D	2	Splitter bars have shaded 3-D style borders

SplitterBarJoinStyle (SSSplitter)

Constant	Value	Description
ssJoinContinuous	0	Splitter bar junctions are seamless.
ssJoinSegmented	1	Splitter bar junctions display borders.

SplitterBarType (SSSplitter)

Constant	Value	Description
ssSplitterBarBoth	0	Splitter bar junctions is being moved.
ssSplitterBarHorizontal	1	Horizontal splitter bar is being moved.
ssSplitterBarVertical	2	Vertical splitter bar is being moved.

SplitterResizeStyle (SSSplitter)

Constant	Value	Description
ssResizeNonProportional	0	Splitter panes will not resize proportionally to control size.
ssResizeProportional	1	Splitter panes will resize proportionally to control size.

SplitType (SSSplitter)

Constant	Value	Description
ssTopOfSplit	0	New pane will be created above existing pane.
ssBottomOfSplit	1	New pane will be created below existing pane.
ssLeftOfSplit	2	New pane will be created to the left of existing pane.
ssRightOfSplit	3	New pane will be created to the right of existing pane.

ActiveThread Controls



Using the ActiveThread Controls

Describes how to use the features that are common to all the ActiveThread controls, including common properties, events and methods.



SSCheck Box Control

Give you multi-state data entry and display functions with a highly customizable appearance.



SSCommand Button Control

Provides you with buttons that can have a variety of styles for use on your forms.



SSFrame Control

Used as a container for other controls. Provides unique background capabilities and active features



SSOption Button Control

Give you exclusive data entry and display functions with a highly customizable appearance.



SSPanel Control

Used as a container for other controls. Provides active features and can also be used as a progress indicator.



SSRibbon Button Control

Gives you buttons with exclusive and group operation functions for use on forms or toolbars.



SSSplitter Control

Organize the controls of your application based on a series of resizable panes of varying sizes, separated by movable splitter bars.

Add Method

[See Also](#)

[Example](#)

[Applies To](#)

Description

This method is used to add a new pane to the Splitter control by splitting an existing pane.

Syntax

```
object.Add panename, splitdir, [newpanename]
```

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>panename</i>	A string expression that evaluates to the name of an existing pane object that will be split to create the new pane.
<i>splitdir</i>	An integer expression specifying the location of the new pane, as described in Settings.
<i>newpanename</i>	Optional. A variant expression specifying a name for the newly created pane.

Settings

Setting	Description
0	Top of split. The pane will be divided horizontally and the new pane will appear above the existing one.
1	Bottom of split. The pane will be divided horizontally and the new pane will appear below the existing one.
2	Left of split. The pane will be divided vertically and the new pane will appear to the left of the existing one.
3	Right of split. The pane will be divided vertically and the new pane will appear to the right of the existing one.

Remarks

The *panename* supplied must be an existing **Pane** object from the **Panes** collection.

When splitting a pane, the existing pane is divided exactly in half in the direction implied by *splitdir* (horizontal for 0 and 1, vertical for 2 and 3.) If you wish you can specify a name for the new pane at the time of creation by supplying a value for *newpanename*. If you do not specify a name, the default name for the new pane will be used.

Once the new pane has been added, you can change its size using the **Height** and **Width** properties of the **Pane** object.

Note You can only split a pane that does not contain a control. If you attempt to split a pane containing a control, an error will occur.

Adding Animation To Controls

You can add animated pictures to your controls in the same way that you add static pictures. Animated pictures are simply regular bitmaps divided into a number of equally sized segments that constitute the "frames" of the animation. Segmented bitmaps are assigned to the **Picture** property of the control, or to one of the associated picture properties (**PictureDisabled**, **PictureDn** or **PictureDnDisabled**; animation cannot be applied to the **PictureBackground** property)

You enable the animated capabilities of a control by setting the **PictureFrames** property of the control to a number greater than one. The number you specify for **PictureFrames** is the number of segments in your bitmap.

The property pages of the ActiveThreed controls contain a special Animation Builder tab that gives you the ability to easily import multiple bitmaps into one segmented bitmap, then apply that bitmap to the **Picture** property of the control. You can also set the number of frames that will be used in the resultant bitmap.

To add an animated picture to your control:

1. Select the **Picture** property of the control and specify the name of a segmented bitmap that will be used as an animation. You can create your own segmented bitmaps, or use one of the ones that comes with ActiveThreed.
2. Enter the number of segments in the bitmap as the value for the **PictureFrames** property.

Note that you can obtain some special animation effects by setting **PictureFrames** to a value other than the actual number of segments in the bitmap. An incorrect setting of **PictureFrames** can result in a "rolling" animation that scrolls across the picture area of the control in addition to exhibiting an animated action. You may want to experiment with different settings for **PictureFrames** to see what kind of effects you can produce.

Adding Pictures to Captions

All of the ActiveThread controls (except the SSSplitter) can incorporate a picture into the area occupied by their caption. The implementation of this feature varies from control to control. In the SSCheck and SSOption controls, the picture shares the caption area with the caption, but text and picture remain separate, and the picture can be aligned only to the right or left of the caption. For the SSCommand, SSRibbon and SSPanel controls, the caption and the picture share the area of the control, and can be aligned with respect to or independently of one another. The SSFrame provides a picture that is restricted to the caption area of the control, similar to that of the SSCheck and SSOption.

You can add a picture to a control by specifying the name of a bitmap, icon or metafile for the control's **Picture** property. Certain controls support multiple pictures, such as the SSCommand which supplies separate picture properties for the down and disabled states of the button via the **PictureDn** and **PictureDisabled** properties. The SSFrame and SSPanel controls have a **PictureBackground** property that lets you specify a picture for the background of the control that is distinct from the control's caption picture as specified by the **Picture** property.

Because you have a number of picture properties (**Picture**, **PictureDn**, **PictureDisabled**, **PictureDnDisabled**) it is possible to set them to use pictures of different sizes. Doing so will result in the dimensions of the largest picture being used as the de facto picture size for the control. This is most apparent when using the **AutoSize** property to adjust the size of the control to the size of the picture.

If you specify a metafile for the **Picture** property, some of the properties of the control relating to picture alignment may behave somewhat differently. When you use a metafile, if the control has a caption, the **AutoSize** property (SSCommand, SSRibbon, SSPanel) has no effect. The effect of the **PictureAlignment** depends on whether the picture is being aligned to the control or to the caption. If the setting of **PictureAlignment** is such that the metafile will be aligned to the control (Top Left, Center Bottom, etc.) the property is ignored and the metafile fills the full area of the control. If however the metafile is aligned to the caption (Left of Caption, Top of Caption, etc.) the metafile will be scaled into an area with the specified alignment. For example, if **PictureAlignment** is set to "Right of Caption" then the metafile will be scaled into whatever area of the control has the specified alignment and is outside the caption area:



To add a picture to your control:

1. Select the **Picture** property of the control
2. Enter the filename of a bitmap, icon or metafile, or choose one from the File Open dialog.
3. Select the type of alignment you wish the picture to have by choosing a value for the **PictureAlignment** property.

To add a background picture to an SSFrame or SSPanel:

1. Select the **PictureBackground** property of the control
2. Enter the filename of a bitmap, icon or metafile, or choose one from the File Open

dialog.

3. Select the **PictureBackgroundStyle** property and choose the way you want the background picture to be aligned; centered, stretched or tiled.

Align Property

[See Also](#)

[Example](#)

[Applies To](#)

Description

Returns or sets a value that determines how the control will be aligned to its container.

Syntax

object.**Align**[= *number*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the alignment of the control, as described in Settings.

Settings

Setting	Description
0	None
1	Top of container
2	Bottom of container
3	Left of container
4	Right of container

Remarks

This is a standard property found on container controls. A control with the **Align** property set to greater than zero will automatically resize itself when its container is resized.

Note When changing the setting of the **Align** property through code, you may have to refresh the control in order for the **AutoSize** property to take effect.

Alignment Property

[See Also](#)

[Example](#)

[Applies To](#)

Description

Returns or sets a value that determines how the caption of the control will be aligned.

Syntax

object.**Alignment**[= *number*]

Part	Description
------	-------------

<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
---------------	---

<i>number</i>	An integer expression specifying the alignment of the caption of the control, as described in Settings.
---------------	---

Settings

For *SSCheck* and *SSOption*

Setting	Description
---------	-------------

0	(Default) Left Justify
---	------------------------

1	Right Justify
---	---------------

For *SSFrame*

Setting	Description
---------	-------------

0	(Default) Top Left
---	--------------------

1	Top Right
---	-----------

2	Top Center
---	------------

3	Bottom Right
---	--------------

4	Bottom Left
---	-------------

5	Bottom Center
---	---------------

For *SSCommand*, *SSPanel* and *SSRibbon*

Setting	Description
---------	-------------

0	(Default - <i>SSPanel</i>) 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 - <i>SSCommand</i> & <i>SSRibbon</i>) Center - Middle
---	---

8	Center - Bottom
---	-----------------

Remarks

For the *SSCheck* and *SSOption*, **Alignment** controls how the check box or option button graphic will be aligned within the control:

<input checked="" type="radio"/> Left Justified SSOption	Right Justified SSOption <input type="radio"/>
<input type="checkbox"/> Left Justified SSCheck	Right Justified SSCheck <input checked="" type="checkbox"/>

For the SSFrame, the **Alignment** property controls the alignment of the entire caption area relative to the control itself. The setting of **Alignment** also affects the placement of any picture specified by the **Picture** property.

For SSCommand, SSPanel & SSRibbon, **Alignment** controls the alignment of the caption text within the entire area of the control. The setting of **Alignment** may or may not affect the alignment and scaling of the control's picture, based on the type of alignment, the type of picture and the type of picture alignment. See [**Adding Pictures To Captions**](#) for more information.

Collection Summary

[See Also](#)

A	B	C	D	E	F	G	H	I	J	K	L	M
N	O	P	Q	R	S	T	U	V	W	X	Y	Z

Panes

ssDataObjectFiles

Event Summary

[See Also](#)

A	B	C	D	E	F	G	H	I	J	K	L	M
N	O	P	Q	R	S	T	U	V	W	X	Y	Z

C

[Click](#)

D

[DbClick](#)

K

[KeyDown](#)

[KeyPress](#)

[KeyUp](#)

M

[MarqueeCycleBegin](#)

[MarqueeCycleEnd](#)

[MouseDown](#)

[MouseEnter](#)

[MouseExit](#)

[MouseMove](#)

[MouseUp](#)

O

[OLECompleteDrag](#)

[OLEDragDrop](#)

[OLEDragOver](#)

[OLEGiveFeedback](#)

[OLESetData](#)

[OLEStartDrag](#)

R

[Resize](#)

S

[SplitterEndDrag](#)

[SplitterStartDrag](#)

Method Summary

[See Also](#)

A	B	C	D	E	F	G	H	I	J	K	L	M
N	O	P	Q	R	S	T	U	V	W	X	Y	Z

[Note *](#)

A

Add

C

Clear

D

DoClick

G

GetData

GetFormat

I

Item

O

OLEDrag

P

PaneFromControl

PaneFromPosition

PaneFromPositionEx

PlaySoundFile

R

Refresh

Remove

S

SetData

SetFocus *

Object Summary

[See Also](#)

A	B	C	D	E	F	G	H	I	J	K	L	M
N	O	P	Q	R	S	T	U	V	W	X	Y	Z

Font

Pane

ssDataObject

Property Summary

[See Also](#)

A	B	C	D	E	F	G	H	I	J	K	L	M
N	O	P	Q	R	S	T	U	V	W	X	Y	Z

[Note *](#)

[\(About\)](#)

[\(Custom\)](#)

A

[Align](#)

[Alignment](#)

[AutoRepeat](#)

[AutoSize \(SSCommand & SSRibbon\)](#)

[AutoSize \(SSPanel\)](#)

[AutoSize \(SSplitter\)](#)

B

[BackColor](#)

[BackStyle](#)

[BevelInner](#)

[BevelOuter](#)

[BevelWidth](#)

[Bold](#)

[BorderStyle](#)

[BorderWidth](#)

[ButtonStyle](#)

C

[Cancel](#)

[Caption](#)

[CaptionStyle](#)

[CheckBoxGraphics](#)

[CheckBoxMaskColor](#)

[CheckBoxUseMask](#)

[ClipControls](#)

[Control](#)

[ControlName](#)

[Count](#)

D

[DataField *](#)

[DataSource *](#)

[Default](#)

[DragMode](#)

E

[Enabled](#)

F

[Files](#)

[FloodColor](#)

FloodFillStyle

FloodPercent

FloodShowPct

FloodType

Font

Font3D

ForeColor

G

GroupAllowAllUp

GroupNumber

H

Height *

HelpContextID *

Hwnd

I

Italic

L

Left *

Locked

LockHeight

LockWidth

M

MarqueeDelay

MarqueeDirection

MarqueeScrollAmount

MarqueeStyle

MinHeight

MinWidth

Mouselcon

MousePointer

N

Name *

Name (Font object)

O

OLEDropMode

OptionBtnGraphics

OptionBtnMaskColor

OptionBtnUseMask

Outline

P

Picture

PictureAlignment

PictureAnimationDelay
PictureAnimationEnabled
PictureBackground
PictureBackgroundStyle
PictureDisabled
PictureDisabledFrames
PictureDn
PictureDnChange
PictureDnDisabled
PictureDnDisabledFrames
PictureDnFrames
PictureFrames
PictureMaskColor
PictureUseMask

R

RoundedCorners

S

ShadowStyle

Size

SplitterBarAppearance

SplitterBarJoinStyle

SplitterBarWidth

SplitterResizeStyle

Strikethrough

T

TagVariant

Top *

TripleState

U

Underline

V

Value (SSCheck)

Value (SSCommand, SSOption, SSRibbon)

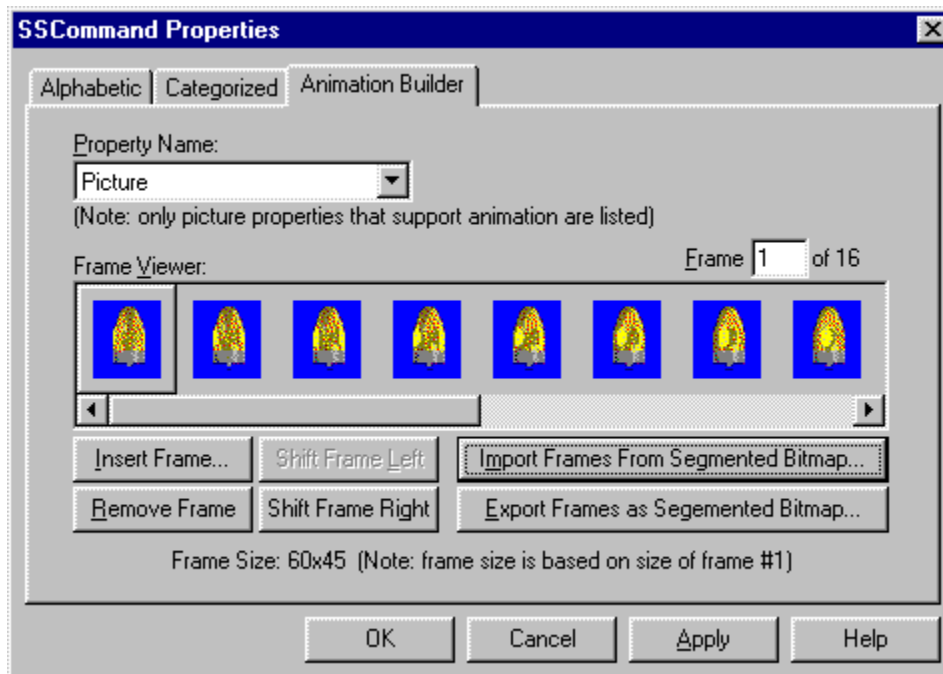
W

Width *

Windowless

Animation Builder Property Page

The SSFrame, SSPanel, SSOption, SSCheck, SSRibbon and SSCommand controls provide an Animation Builder property page. Here is what it looks like:



The Animation Builder page enables you to setup a sequence of bitmaps that can subsequently be animated by ActiveThread controls.

When you click "Insert Frame..." to add individual files to a segmented bitmap, the File Open dialog gives you the ability to select multiple files. The order in which these files are selected is significant. The last file you select will be the first file imported into the segmented bitmap. Because the files are imported in reverse order, you should begin by selecting the last file you want to use, and work your way back to the first one. For example, if you wish to import ten bitmaps, BMP01.BMP through BMP10.BMP, you should select BMP10.BMP first, then BMP09.BMP and so on until you finally select BMP01.BMP.

Creating a Bitmap Sequence

To create a bitmap sequence from a set of bitmaps:

1. Create or acquire the sequence of bitmaps that are to be animated.
2. Select a picture-type property of the control in the Property Name field.

Note Only 'Picturexxx' properties that have an associated 'PicturexxxFrames' property are available. Not all picture-related properties are available for all ActiveThread controls.

3. Press the Insert Picture button and select a bitmap using the Insert Picture dialog.

At this point a preview of the bitmap will appear in the sequence displayed in the box labeled Frames.

Removing a Bitmap from the Sequence

To remove a bitmap from the sequence displayed in the box labeled Frames:

1. Select the bitmap you want to remove by directly clicking on it in the Frames box.
2. Press the Remove Picture button.

At this point the bitmap will be removed from the sequence.

AutoRepeat Property

[Applies To](#) [Example](#)

Description

Returns or sets a value that determines whether the control will automatically generate multiple click events.

Syntax

object.**AutoRepeat**[= *boolean*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying the repeat mode of the control, as described in Settings.

Settings

Setting	Description
True	The button will generate multiple Click events when the mouse button is held down over the control, or the spacebar is held down while the control has focus.
False	(Default) The button will generate a single Click event when the mouse button is held down over the control, or the spacebar is held down while the control has focus.

Remarks

Use the **AutoRepeat** property when you want the user to be able to perform an action repeatedly. The button will continue to fire the **Click** event as long as the mouse cursor is over the control and the user is holding down the mouse button.

There is an initial delay of 500 milliseconds between the time the user clicks the button and the time the AutoRepeat function takes over. Once the button has begun to AutoRepeat, additional **Click** events will be fired at 100 millisecond intervals. (Times are approximate and may vary depending on the level of overall system activity.)

AutoSize Property (SSCommand & SSRibbon)

[See Also](#)

[Applies To](#)

Description

Returns or sets a value that specifies how the control will adjust it's size based on the size of the picture.

Syntax

object.**AutoSize** [= *number*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying how the control will resize itself, as described in Settings.

Settings

Setting	Description
0	(Default) None. The control will not resize itself.
1	Button To Picture. The control will resize itself to match the dimensions of the picture specified in the Picture property.
2	Picture to Button. The control will resize the picture specified in the Picture property to match the dimensions of the control.

Remarks

This property is used primarily to create custom button effects. When **AutoSize** is set to 1, the button will automatically assume the dimensions of the picture specified by the **Picture** property. By setting the **ButtonStyle** property to 'Borderless' and using mask colors in the picture, you can create a wide variety of effects, from irregularly shaped buttons to buttons that seem to float over their container. Using the **PictureDn**, **PictureDisabled**, and for the Ribbon button, **PictureDnDisabled** properties, you can control the appearance of the button in any of its states.

Note This property has no effect when a caption is specified for the control.

Although you can use **AutoSize** to create buttons that appear irregularly shaped, the active area of the button is always rectangular.

Since you can specify pictures of different sizes for the **Picture**, **PictureDn**, **PictureDisabled**, and **PictureDnDisabled** properties, if you set **AutoSize** to '1 - Button To Picture', the control will adjust itself based on the size of the largest picture specified in any of these properties. For example, if the picture specified for **PictureDisabled** is larger than any of the other pictures, **PictureDisabled** will be used to adjust the size of the control.

AutoSize Property (SSPanel)

[See Also](#)

[Applies To](#)

Description

Returns or sets a value that determines whether the control will resize itself automatically based on the size of the caption.

Syntax

object.**AutoSize** [= *number*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying how the control will resize itself, as described in Settings.

Settings

Setting	Description
0	None
1	Width to Caption
2	Height to Caption
3	Child to Panel

Remarks

Use the **AutoSize** property to have the control automatically adjust its size based on the size of the caption. You can also use this property to have the control automatically resize it's child control to fill the area of the panel. This setting will only take effect if there is a single child control on the panel.

AutoSize Property (SSSplitter)

[See Also](#)

[Applies To](#)

Description

Returns or sets a value that determines whether the control will automatically resize itself to fill its container.

Syntax

object.**AutoSize** [= *number*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying how the control will resize itself, as described in Settings.

Settings

Setting	Description
0	(Default) None
1	Fill Container

Remarks

Use the **AutoSize** property to have the control automatically adjust its size based on the size of its container. This property is similar to the standard **Align** property, except that it aligns the control to all sides of the container at once.

The **AutoSize** property operates exclusively of the **Align** property. If the **Align** property of the SSSplitter is set to anything other than '0 - None' then the **AutoSize** property has no effect.

Note When changing the setting of the **Align** property through code, you may have to refresh the control in order for the **AutoSize** property to take effect.

BackColor Property

[See Also](#)

[Applies To](#)

Description

Returns or sets the background color of the control.

Syntax

object.**BackColor** [= *color*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>color</i>	A value or constant that determines the color of the specified object.

Remarks

The appearance of the control's background is determined by the **BackStyle** property. If **BackStyle** is set to '0 - Opaque', the background of the control will be filled with the color specified by the **BackColor** property. Any other setting for **BackStyle** will cause the background to be filled with something other than the **BackColor**.

BackStyle Property

[See Also](#)

[Applies To](#)

Description

Returns or sets a value that determines whether the background of the control will be opaque or transparent.

Syntax

object.**BackStyle** [= *integer*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>integer</i>	An integer expression specifying the type of background style to use, as described in Settings.

Settings

Setting	Description
0	Opaque. The area of the control will be filled with the control's BackColor .
1	Transparent. The area of the control will allow what is behind the control to show through.

Remarks

The new Transparent style of ActiveThread controls allows the background to show through the control. The SSFrame or SSPanel appear only as a border (with caption and caption picture visible, if specified) giving you a visible and functional container that does not interfere with the background pattern or picture of the application. SSCommand and SSRibbon buttons can also be made transparent, with the button's appearance being dependent on the background of the container, the setting of the **BorderStyle** property, and the settings of any picture-related properties.

If you plan to use a transparent background for any control, there are certain procedures you must observe. The transparent control's container, whether it be a form or another container control, must have its **ClipControls** property set to False. Also, any control that is to use a transparent background must be set to have a transparent **BackStyle** at design time. If a control is transparent at design time, you may change between transparent and opaque modes at run time. However, if a control is opaque at design time, and you attempt to set to transparent at run time, an error will occur.

BevelInner Property

[See Also](#)

[Applies To](#)

Description

Returns or sets a value that specifies the type of inside beveling for the panel.

Syntax

object.**BevelInner** [= *number*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the type of inner bevel to use.

Settings

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

Remarks

This property gives you control over the three-dimensional appearance of the control's border. This property has no effect if **BorderWidth** is set to 0.

BevelOuter Property

[See Also](#)

[Applies To](#)

Description

Returns or sets a value that specifies the type of outside beveling for the panel.

Syntax

object.**BevelOuter** [= *number*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>Number</i>	An integer expression specifying the type of outer bevel to use.

Settings

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

Remarks

This property gives you control over the three-dimensional appearance of the control's border. This property has no effect if **BorderWidth** is set to 0.

BevelWidth Property

[See Also](#)

[Applies To](#)

Description

Returns or sets the width (in pixels) of the beveled area of the control, which determines the amount of the 3-D effect.

Syntax

object.**BevelWidth** [= *number*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>Number</i>	An integer expression specifying the width of the beveling.

Remarks

This property only applies to controls with Windows 3.X style borders. For other border settings this property has no effect.

The valid range for this property is 0 to 10. The default value for this property is 1.

Bold Property

[See Also](#)

[Applies To](#)

Description

Returns or sets the font style of the specified **Font** object to either bold or non-bold.

Syntax

object.**Bold** [= *boolean*]

The **Bold** property syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>Boolean</i>	A Boolean expression specifying the font style, as described in Settings.

Settings

The settings for *boolean* are:

Setting	Description
True	Turns on bold formatting.
False	Turns off bold formatting.

Remarks

The **Font** object is not directly available at design time. Instead you set the **Bold** property through a control's **Font** property.

At run time, however, you can set **Bold** directly by specifying its setting for the appropriate **Font** object.

BorderStyle Property

[See Also](#)

[Applies To](#)

Description

Returns or sets a value that specifies the type of border for the control

Syntax

object.**BorderStyle** [= *number*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the appearance of the control border, as described in Settings.

Settings

Setting	Description
0	None. No border is drawn.
1	Fixed Single. A single pixel border is drawn.
2	(Default) Inset. The border appears as if it is inset into the background.
3	Raised. The border appears as if it is raised from the background.

Remarks

A setting of 0 (no border) is useful when setting the **AutoSize** property of the control so that it fills the form or container on which it is placed.

BorderWidth Property

[See Also](#)

[Applies To](#)

Description

Returns or sets the width (in pixels) of the space between the outer and inner bevels.

Syntax

object.**BorderWidth** [= *number*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the width of the panel border.

Remarks

If this property is set to 0, the settings of **BevelInner** and **BevelOuter** have no effect.

The valid range for this property is 0 to 30.

ButtonStyle Property

[See Also](#)

[Applies To](#)

Description

Returns or sets a value that specifies the appearance of the button.

Syntax

object.**ButtonStyle** [= *number*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the style of the button, as described in Settings.

Settings

Setting	Description
0	(Default) Button style is operating system dependent. (i.e. Windows 95 or Windows 3.X, depending on the environment.)
1	Windows 95 style.
2	Windows 3.X style.
3	Active Borders style (mouse sensitive borders.)
4	Borderless.

Remarks

The settings of this property provide a variety of interface styles for the buttons of your application. By default, the buttons will adjust themselves to match the overall style of the operating environment, however you can specify specific styles if you wish.

When the Active Borders style is specified for a button, the button appears to have no borders until the user passes the mouse over the button. The borders then appear, and remain visible until the mouse pointer leaves the control.

The SSSCommand button will always display a focus rectangle when it receives focus, whatever the setting of **ButtonStyle**. Ribbon buttons will not display a focus rectangle.

Cancel Property

[See Also](#)

[Applies To](#)

Description

Returns or sets a value that determines whether a command button is the Cancel button on a form.

Syntax

object.**Cancel** [= *boolean*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying whether the object is the Cancel button, as described in Settings.

Settings

Setting	Description
True	The command button is the Cancel button.
False	(Default) The command button is not the Cancel button.

Remarks

This is an [extender](#) property.

Use the **Cancel** property to give the user the option of canceling uncommitted changes and returning the form to its previous state.

Only one command button on a form can be the Cancel button. When the **Cancel** property is set to True for one command button, it's automatically set to False for all other command buttons on the form.

When a command button's **Cancel** property setting is True and the form is the active form, the user can choose the command button by clicking it, pressing the ESC key, or pressing ENTER when the button has the focus..

Caption Property

[See Also](#)

[Applies To](#)

Description

Returns or sets the caption text of the control.

Syntax

object.**Caption** [= *text*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>text</i>	A string expression that evaluates to the text displayed as the caption.

Remarks

The appearance of the caption is based on the **Font** property and the **Alignment** property. The caption may optionally be wrapped using the **CaptionStyle** property or animated using the **MarqueeStyle** property. The alignment of the caption may be affected by any pictures that share the caption area.

CaptionStyle Property

[See Also](#)

[Applies To](#)

Description

Returns or sets a value that specifies how caption text will be displayed on the control.

Syntax

object.**CaptionStyle** [= *number*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the caption style, as described in Settings.

Settings

Setting	Description
0	(Default) Standard. The caption displays as static text on a single line.
1	Wrapped. The caption displays as static text on multiple lines.

Remarks

ActiveThread controls now include a set of active caption styles. You can use these styles to give added impact to your application, or to call attention to particular areas of interest within your program. Caption text is displayed in the *caption area* of the control; the size of this area is generally the size of the control, but may vary based on the specific control being used and on whether or not caption pictures are being displayed.

The **CaptionStyle** property specifies whether caption text will appear on one line or be wrapped to multiple lines. Animated text effects are provided by setting the **MarqueeStyle** property and the **MarqueeDirection** property..

The **CaptionStyle** property only takes effect if **MarqueeStyle** is set to '0 - None' or '3 - Blinking.' Otherwise, text is wrapped automatically as required by the type of marquee animation.

CheckBoxGraphics Property

[See Also](#)

[Applies To](#)

Description

Returns or sets the image to be used for the check box portion of the control.

Syntax


object.**CheckBoxGraphics** [= *picture*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>picture</i>	A picture object specifying a graphic, as described in Settings.

Settings

Setting	Description
(None)	(Default) No picture.
(Bitmap)	Specifies a graphic. You can load the graphic from the properties window at design time. At run time, you can set this property using the LoadPicture function on a bitmap.

Remarks

You can specify a custom appearance for the check box portion of the control using this property, which will replace the standard check box graphic . This is useful for designing a distinctive user interface.

The picture specified for this property is a segmented bitmap containing all the possible states of the control. Several sample bitmaps are included with ActiveThread to get you started in designing your own check box multiple-state bitmaps.

The SSCheck control has nine states. If you are creating custom graphics, your bitmap should have nine segments of equal width. These nine states should appear from left to right in the segmented bitmap. The states of the control are:

0. **Unchecked** - the normal state of the control when unchecked
1. **Checked** - the normal state of the control when checked
2. **Grayed** - the normal state of the control when in an indeterminate state
3. **Unchecked Pressed** - unchecked, with the left mouse button being pressed while over the control
4. **Checked Pressed** - checked, with the left mouse button being pressed while over the control
5. **Grayed Pressed** - indeterminate, with the left mouse button being pressed while over the control
6. **Unchecked Disabled** - the disabled state of the control when unchecked
7. **Checked Disabled** - the disabled state of the control when checked
8. **Grayed Disabled** - the disabled state of the control when indeterminate

You can specify part of the check box graphic as transparent, using the **CheckBoxMaskColor** and **CheckBoxUseMask** properties.

CheckBoxMaskColor Property

[See Also](#)

[Applies To](#)

Description


Returns or sets the color that will become the transparent part of the check box graphic.

Syntax

object.**CheckBoxMaskColor** [= *color*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>color</i>	A value or constant that determines the mask color of the check box graphic.

Remarks

This setting only takes effect when **CheckBoxUseMask** is set to True. This property works in conjunction with the **CheckBoxGraphics** property to create a custom graphic for the check box that replaces the standard check box graphic .

By specifying one of the colors used in the segmented bitmap as the **CheckBoxMaskColor**, you cause that color to become transparent when used by the control. This gives you the ability to design check box graphics that have shapes other than square, or that have transparent areas.

CheckBoxUseMask Property

[See Also](#)

[Applies To](#)

Description

Returns or sets a value that determines whether the control will use the **CheckBoxMaskColor** to create transparent areas.

Syntax

object.**CheckBoxUseMask** [= *boolean*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying whether the mask color will be used, as described in Settings.

Settings

Setting	Description
True	The mask color will be used.
False	(Default) The mask color will not be used.

Remarks

This property enables the use of a mask color in the segmented bitmap specified by the **CheckBoxGraphics** property. When set to True, the color specified by the **CheckBoxMaskColor** property will become transparent in the control. When set to False, all colors in the segmented bitmap will be opaque and visible.

For more information on creating custom check box graphics, see the **CheckBoxGraphics** property.

Clear Method

Applies To

Description

This method deletes the contents of the **ssDataObject** object.

Syntax

object.**Clear**

Part	Description
-------------	--------------------

<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list..
---------------	--

Remarks

This method is available only for component drag sources. If **Clear** is called from a component drop target, an error is generated.

Click Event

[See Also](#)

[Applies To](#)

Description

Occurs when the user clicks the left mouse button and releases it over the control or, in the case of the *SSCommand* and *SSCheck* controls, when the user presses the spacebar while the control has focus.

Syntax (*SSCheck*, *SSOption*, *SSRibbon*)

Sub *control_Click* ([*index As Integer*] *value As Integer*)

The event parameters are:

Parameter Description

<i>index</i>	An integer expression that uniquely identifies a control if it is in a control array.
<i>value</i>	An integer expression that specifies the value of the control.

Syntax (*SSCommand*, *SSPanel*, *SSSplitter*)

Sub *control_Click* ([*index As Integer*])

The event parameters are:

Parameter Description

<i>index</i>	An integer expression that uniquely identifies a control if it is in a control array.
--------------	---

Remarks

The *value* parameter is included for backwards compatibility with previous versions of THREED. You should use the **Value** property of the control in your code, rather than the *value* parameter of this event.

Clicking a control generates **MouseDown** and **MouseUp** events in addition to the **Click** event. The order in which these three events occur varies from control to control. See the Event Maps for further details.

Note To distinguish between the left, right, and middle mouse buttons, use the **MouseDown** and **MouseUp** events.

ClipControls Property

Applies To

Description

Returns or sets a value that determines whether graphics methods in Paint events repaint the entire object or only newly exposed areas. Also determines whether the Microsoft Windows operating environment creates a clipping region that excludes nongraphical controls contained by the object.

Syntax

object.**ClipControls** [= *boolean*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A Boolean expression that specifies how objects are repainted, as described in Settings.

Settings

Setting	Description
True	(Default) Graphics methods in Paint events repaint the entire object. A clipping region is created around nongraphical controls on the form before a Paint event.
False	Graphics methods in Paint events repaint only newly exposed areas. A clipping region isn't created around nongraphical controls before a Paint event. Complex forms usually load and repaint faster when ClipControls is set to False.

Remarks

Clipping is the process of determining which parts of a form or container, such as an SSFrame, SSPanel or SSSplitter control, are painted when the form is displayed. An outline of the form and controls is created in memory. The Windows operating environment uses this outline to paint some parts, such as the background, without affecting other parts, such as the contents of a TextBox control. Because the clipping region is created in memory, setting this property to False can reduce the time needed to paint or repaint a form.

ClipControls must be set to False on the container of a transparent ActiveThread control. Failure to do so will cause the transparent control to repaint improperly.

Avoid nesting controls with **ClipControls** set to True inside a control with **ClipControls** set to False (for instance, an SSPanel inside an SSSplitter). This kind of control nesting causes the controls to repaint incorrectly. To fix this problem, set the **ClipControls** property for both the container control and the nested controls to True.

Compatibility Issues

[See Also](#)

An Introduction to ActiveX Controls

What is an ActiveX control?

An ActiveX (or OCX) control is a specific type of DLL that makes use of *OLE Automation* to provide functions to other programs. An ActiveX control is an in-process *OLE server*, and the program that uses its services is an *OLE client*. ActiveX controls can provide a nearly unlimited range of functions to their clients.

How is an ActiveX control different from a VBX control?

The VBX control specification was designed exclusively for use with Visual Basic. Although some other languages offer limited VBX support, the majority of VBX controls function only in Visual Basic. VBX controls are also limited in other ways. Their 16-bit architecture restricts their ability to use memory and to function in a 32-bit operating system, such as Windows NT.

The difference between ActiveX and VBX controls may not even be apparent to you if you program exclusively in Visual Basic. You access the properties of an ActiveX control at design time and through code just as you do the properties of a VBX. The process of including both types of controls in your project and distributing them is very similar. The similarities end when you move outside of the Visual Basic programming environment.

ActiveX controls are supported by a much wider range of host environments, including other languages, database management systems, and productivity applications. ActiveX controls can be used as the building blocks in a modular software environment, where a complete project might include your own code, custom controls and commercial applications all working together. ActiveX controls also have the ability to make full use of the newest 32-bit operating systems, taking advantage of improved memory access, better multi-tasking and increased performance.

See Also

[Upgrade Notes](#)

Contacting Technical Support

Internet

You can send electronic mail to technical support via the Internet. Messages should be addressed to: support.3D@shersoft.com

For up-to-the-minute information and the latest updates, as well as general information about Sheridan Software Systems Inc. and our products, visit our home page on the World Wide Web. The address is <http://www.shersoft.com>

CompuServe

You can obtain technical support on CompuServe by visiting the SHERIDAN forum. You can type [GO SHERIDAN](#) at any CompuServe prompt. Or you can send e-mail to our technical support department at [74774,547](#)

FAX

To fax questions or comments regarding any Sheridan product, dial [\(516\) 753-3661](#).

Telephone Support

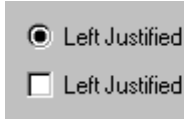
For technical support for this or any other Sheridan product, contact Sheridan Software systems at [\(516\) 753-0985](#). You can either speak to a live technical support representative or get answers using the Automated Fax Service.

Sheridan's support hours are 9AM to 5PM (EST), Monday through Friday.



[Copyright Notice](#)

[Credits](#)



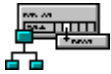
ActiveThread Controls

A reference guide to the controls, with summaries of how to use their common and individual features. Also includes lists of the properties, methods and events found in each control.



Quick Tours

Step-by-step introductions to using the ActiveThread controls and the Splitter control.



Control Reference

An alphabetical listing of all programming language topics

Properties

Events

Methods

Objects

Collections

Errors

Constants



Technical Specifications

A list of system requirements, included files and troubleshooting tips.



Contacting Technical Support

How to obtain technical and product support for Sheridan products.

Control Background Effects

ActiveThread controls provide a number of different background effects that can add style to your application. The range of effects available depends on the control.

All controls (except the SSSplitter) support some kind of picture in addition to the caption text. The two container controls (SSFrame and SSPanel) support two types of pictures; a background picture and a caption picture. The background picture can be stretched or tiled to fill the area of the control. The rest of the controls support a caption picture only.

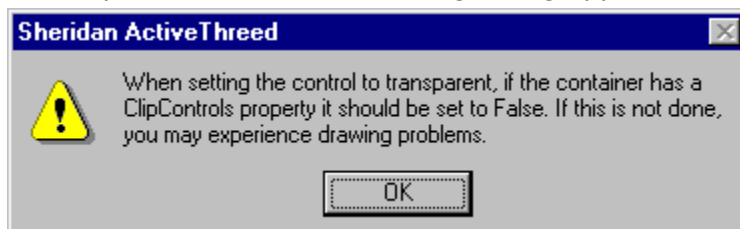
In addition to pictures, all of the ActiveThread controls support transparent backgrounds. It is also possible to add transparency to your pictures, whether the picture is in the background or in the caption.

Transparent Operation

Any of the ActiveThread controls may have a transparent background. To make the control transparent, simply set the **BackStyle** property to '1 - Transparent.' However, when using transparent controls, you must set the **ClipControls** property of the control's container (if it has one) to False. The first time you create a transparent control in a project, you will see a warning message to this effect. If you fail to do this, the control may not redraw itself properly.

To make a control transparent:

1. Select the control's container, and verify whether or not it has a **ClipControls** property. If it does, set the property value to False.
2. Select the control you wish to make transparent.
3. Select the **BackStyle** property of the control and change its value to '1 - Transparent'.
4. If this is the first transparent control you have created in a project, you will need to press OK when the warning dialog appears:



Note If you wish to make a control transparent at run time, it must first be set to transparent at design time. If you attempt to set a non-transparent control to transparent at run-time, an error occurs.

Control Property

[See Also](#)

[Example](#)

[Applies To](#)

Description

Returns or sets the control or the handle to the control located on the pane.

Syntax

object.**Control** [= *control*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>control</i>	A variant expression that evaluates to the control or the window handle of a control located on a Splitter pane.

Remarks

You can use the **Control** property to access any of the properties of the control located on the specified pane.

In environments that support it, such as Visual Basic, the value returned by **Control** is a *dispatch pointer* to the control itself. The dispatch pointer is a stand-in for the control that can be used syntactically in the same way as the control itself. For example, to change the background color of Command button "SSCommand1" on the first pane of a Splitter control, you would use the following code:

```
SSSplitter1.Panes(0).Control.BackColor = &H00C0C0C0&
```

This is functionally identical to the following:

```
SSCommand1.BackColor = &H00C0C0C0&
```

In environments that do not support dispatch pointers, such as Delphi, the value returned by **Control** is the window handle of the control.

The **Control** property is also used to dynamically move controls among panes at run time, as well as moving controls into or out of the Splitter control. If a pane contains a control, and you set the **Control** property of the pane to a second control, the second control will move into the pane, and the original control will be moved into an empty pane, if one is available. If there are no empty panes available, what happens to the original control depends on where the second control came from.

If the second control is coming from another pane in the same Splitter, the two controls will switch places. If the second control is coming from outside the Splitter, the original control will be removed from the Splitter and re-parented to the container of the Splitter control. Similarly, if you set the **Control** property of a **Pane** object to **Null** in code, the control on that pane is automatically re-parented to the Splitter's container, and the pane becomes empty.

ControlName Property

[See Also](#)

[Example](#)

[Applies To](#)

Description

Returns the name of the control located on the pane.

Syntax

object.**ControlName** [= *name*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>name</i>	A string expression that evaluates to the name of a control located on a Splitter pane.

Remarks

This property is read-only.

In environments that support it, such as Visual Basic, the value returned by **ControlName** is the value of the **Name** property of the control, as obtained by the *dispatch pointer* to the control. In environments that do not support dispatch pointers, such as Delphi, the **Name** property is not available at run-time. Therefore, the control's **Caption** is returned.

Copyright © 1997 Sheridan Software Systems, Inc. All rights reserved.

Information in this document is subject to change without notice and does not represent a commitment on the part of Sheridan Software Systems, Inc. The software described in this document is furnished under a license agreement or nondisclosure agreement. The software may be used or copied only in accordance with the terms of the agreement. It is against the law to copy the software on any medium except as specifically allowed in the license or nondisclosure agreement. No part of this manual may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, for any purpose without the express written permission of Sheridan Software Systems, Inc.

ActiveThread, the ActiveThread Logo, and the Sheridan logo are trademarks of Sheridan Software Systems, Inc.

Microsoft, Visual Basic, and Windows are registered trademarks of Microsoft Corporation. All other trademarks and registered trademarks are the property of their respective owners.

This help file was produced using Microsoft Word for Windows from Microsoft Corporation and ForeHelp from ForeFront, Incorporated.

Count Property

[See Also](#)

[Applies To](#)

Description

Returns the total number of panes in the collection.

Syntax

object.**Count** [= *number*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the number of items in the collection.

Remarks

Use this property to find the total number of panes that have been created in a Splitter control.

Product Manager

BradP

Programming

NickC
KrishC
VladL

Documentation

JimD
RajM

Quality Assurance

JohnC
GaryD
SaroK
JasonM

**ActiveThread
Development Team**

DONE

The *Sheridan* ActiveThread Development Team

Product Manager

BradP

Programming

KrishC

NickC

VladL

Documentation

JimD

RajM

Quality Assurance

JohnC

GaryD

SaroK

JasonM

DONE

DataField Property

[See Also](#)

[Applies To](#)

Description

Returns or sets a value that binds the control to a field in the current database record.

Syntax

object.**DataField** [= *name*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>name</i>	A string expression that evaluates to the name of one of the fields in the recordset specified by the DataSource property, to which the value of the control will be bound.

Remarks

This is an [extender](#) property.

This property is environment-dependent. In environments that provide data binding, you can use the **DataSource** property to specify a source of data records, such as the Visual Basic Data Control, then use the **DataField** property to bind the control to a particular field within the recordset. This property may not be available in all environments.

The value that is retrieved or stored in the database by the control depends on the control being used. The SSCheck control will display boolean values from the database, and changing the value of the control will cause the value in the database to change when the recordset is updated. The SSPanel control can display any text or numeric data from the database.

The SSPanel can display data automatically via its **Caption** property, but will only change the stored value if the control's caption is changed through code and the record is updated.

Consult the documentation for your development environment for more information on using data binding features.

DataSource Property

[See Also](#)

[Applies To](#)

Description

Returns or sets the name of the data control used to bind the control at run time.

Syntax

object.**DataSource** [= *name*]

The **DataSource** property syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>name</i>	A string expression that evaluates to the name of the Data control to which the control will be bound at run time.

Remarks

This is an [extender](#) property.

This property is environment-dependent. In environments that provide data binding, you can use the **DataSource** property to specify a source of data records, such as the Visual Basic Data Control, then use the **DataField** property to bind the control to a particular field within the recordset. This property may not be available in all environments.

Unlike the **DataField** property, the **DataSource** property setting is not available at run time.

Consult the documentation for your development environment for more information on using data binding features.

DbIClick Event

[See Also](#)

[Applies To](#)

Description

Occurs when the user double-clicks the mouse while the mouse pointer is over a splitter bar.

Syntax

Sub *control_DbIClick* ([*index As Integer*])

The event parameters are:

Parameter Description

<i>index</i>	An integer expression that uniquely identifies a control if it is in a control array.
--------------	---

Remarks

Clicking a control generates **MouseDown** and **MouseUp** events in addition to the **Click** event.

If **DbIClick** doesn't occur within the system's double-click time limit, the object recognizes another **Click** event. The double-click time limit may vary because the user can set the double-click speed in the Windows Control Panel.

Note To distinguish between the left, right, and middle mouse buttons, use the **MouseDown** and **MouseUp** events.

Default Property

[See Also](#)

[Applies To](#)

Description

Returns or sets a value that determines whether a command button is the default button on a form.

Syntax

object.**Default** [= *boolean*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying whether the object is the Default button, as described in Settings.

Settings

Setting	Description
True	The command button is the Default button.
False	(Default) The command button is not the Default button.

Remarks

This is an [extender](#) property.

Only one command button on a form can be the default command button. When **Default** is set to True for one command button, it's automatically set to False for all other command buttons on the form.

When the command button's **Default** property setting is True and its parent form is active, the user can choose the command button (invoking its **Click** event) by pressing ENTER. Any other control with the focus doesn't receive a keyboard event (KeyDown, KeyPress, or KeyUp) for the ENTER key unless the user has moved the focus to another command button on the same form. In this case, pressing ENTER chooses the command button that has the focus instead of the default command button.

Distributable Files

This section describes files you will need to distribute in addition to your application files and runtime DLL's.

If your application makes use of ActiveThreed controls (THREED20.OCX or SPLITTER.OCX), you will need to install the following files on the user's system:

File	Description
THREED20.OCX	ActiveThreed file that contains the SSCheck, SSCommand, SSFrame, SSOption, SSPanel, and SSRibbon ActiveX controls. This file should be distributed with applications that contain any of the controls mentioned above.
SPLITTER.OCX	ActiveThreed file that contains the SSSplitter ActiveX control. This file should be distributed with applications that contain the SSSplitter control.

In addition, due to the nature of the OLE architecture, the new ActiveX controls require the following support files to be shipped with your application. Version numbers of the files you distribute should be equal to or greater than those listed here:

ASYCFILT.DLL	2.20.4054
MSVCRT.DLL	4.20.6201
OLEAUT32.DLL	2.20.4049
OLEPRO32.DLL	5.0.4055
STDOLE2.TLB	2.20.4049

Distribution Notes

Distributable Files

Non-Distributable Files

DoClick Method

[See Also](#)

[Example](#)

[Applies To](#)

Description

This method is used to trigger the **Click** event of the control.

Syntax

object.**DoClick**

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list..

Remarks

Invoking this method causes the **Click** event of the control to be fired and any code contained in it to be executed.

DragMode Property

[See Also](#)

[Applies To](#)

Description

Returns or sets a value that determines whether manual or automatic drag mode is used for a drag-and-drop operation.

Syntax

object.**DragMode** [= *number*]

The DragMode property syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to an object in the Applies To list.
<i>number</i>	An integer expression that specifies the drag mode, as described in Settings.

Settings

The settings for number are:

Setting	Description
0	(Default) Manual - requires using the Drag method to initiate a drag-and-drop operation on the source control.
1	Automatic - clicking the source control automatically initiates a drag-and-drop operation. OLE container controls are automatically dragged only when they don't have the focus.

Remarks

When **DragMode** is set to 1 (Automatic), the control doesn't respond as usual to mouse events. Use the 0 (Manual) setting to determine when a drag-and-drop operation begins or ends; you can use this setting to initiate a drag-and-drop operation in response to a keyboard or menu command or to enable a source control to recognize a **MouseDown** event prior to a drag-and-drop operation.

Also, when **DragMode** is set to automatic, animations are disabled for the control.

Note While a control is being dragged, it can't recognize other user-initiated mouse or keyboard events (**KeyDown**, **KeyPress** or **KeyUp**, **MouseDown**, **MouseMove**, or **MouseUp**).

Enabled Property

Applies To

Description

Returns or sets a value that determines whether the object can be selected by the user.

Syntax

object.**Enabled** [= *boolean*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying the state of the object, as described in Settings.

Settings

Setting	Description
True	(Default) The object can be selected by the user using the mouse or keyboard.
False	The object cannot be selected

Remarks

Changing the **Enabled** property of an object changes its appearance, and may cause a different picture or graphic to be displayed.

Exercise 10: Adding A Custom Control Graphic

ActiveThreed controls give you the power to customize your applications in many ways. You have already seen how to create custom-designed buttons using the `SSCommand` and `SSRibbon` buttons. Similarly, you can customize the working part of the `SSCheck` and `SSOption` controls. You can replace the standard square check box and round option button with graphics of your own design.

Note Files covering all the concepts in this exercise, including any code you need to type, can be found in the Visual Basic project `EX10.VBP`, located under `QUICK TOUR 1` in the `SAMPLES` subdirectory of your installation directory. The indicated picture files may also be found in this directory.

1. Create a new project, and place an `SSCheck` control on `Form1`.
2. To replace the existing control states, you assign a custom bitmap to the **CheckBoxGraphics** property. This must be a segmented bitmap, similar to the ones used to provide animation, but it can only contain nine segments. Each segment corresponds to one of the states of the control. The `SSCheck` control has the following nine states, and the segments must appear from left to right in this order:
 1. **Unchecked**
 2. **Checked**
 3. **Grayed**
 4. **Unchecked Pressed**
 5. **Checked Pressed**
 6. **Grayed Pressed**
 7. **Unchecked Disabled**
 8. **Checked Disabled**
 9. **Grayed Disabled**

The first three segments provide the normal states of the control. The "Pressed" segments represent how the control will look when the mouse is being clicked on the control. The final three segments define the control's appearance when it is disabled.

ActiveThreed ships with a number of segmented check box graphics to get you started in designing your own. For now, set **CheckBoxGraphics** to `HCHECK.BMP`. This is a check box graphic that gives the check box an informal, handwritten appearance.

3. As with control pictures and animations, check box graphics can use mask colors to create transparent areas. Set the **CheckBoxMaskColor** to white. You can select it from the property sheet color palette, or enter the value `&H00FFFFFF&` for the property.
4. As with the standard picture properties, once you have selected the correct mask color, you must turn on masking in order to activate the transparent effect. Do this by setting **CheckBoxUseMask** to `True`.
5. Run the project and click the check box a few times. The custom graphic replicates the functions of the default check box control, but with a different appearance. Stop the project when you are done.
6. Add an `SSOption` button to your form. The Option button's **OptionBtnGraphics** property uses a segmented graphic for customization as well, but the option

button graphic has only six segments. From left to right, they are:


- 1. Unselected**
- 2. Selected**
- 3. Unselected Pressed**
- 4. Selected Pressed**
- 5. Unselected Disabled**
- 6. Selected Disabled**

Set **OptionBtnGraphics** to HOPTION.BMP, the supplied option button graphic.

7. To make the graphic transparent, set the **OptionBtnMaskColor** property to white. You can select it from the property sheet color palette, or enter the value &H00FFFFFF& for the property.
8. Enable the use of mask colors by setting the **OptionBtnUseMask** property to True.
9. Once you have set up the option button to use the custom graphic, select it on the form and copy it to the clipboard. Then paste it onto the form twice. Answer No when asked if you want to create a control array. This will give you three SSOption buttons with the same custom graphic.
10. Run the project, and click on each of the option buttons. When you have seen enough, stop the project.

By now, changing the appearance of ActiveThreed controls should be second nature. You have seen that not only can you add decorative pictures to your check and option controls, you can change the appearance of the control itself, creating an entirely custom interface that suits the style of your application. The technique for changing a control's appearance is very similar to that for adding a picture.

The next exercise covers a unique feature of the SSCheck control, and discusses some issues of compatibility with existing Windows common controls and previous versions of THREED.

Next Exercise 

Exercise 11: Working With Control States

Previous versions of the THREED controls provided a check box with two logical states - checked and unchecked. These two states are sufficient to perform most of the operations for which check boxes are used. However, the Windows common check box control supports a third state - grayed. Grayed is an indeterminate state that is neither checked nor unchecked. It is most commonly used when the state of a check box is affected by the states of other controls in the application.

You are probably familiar with this concept from installing software. Often an installation program will group the software's component pieces into categories. You have the option to install just the categories or components you want by selecting check boxes. If you choose a category, check off some of its components and uncheck others, the check box for the category as a whole becomes grayed.

The SSCheck control in ActiveThread supports both types of operation. If you are using a previous version of THREED, you do not need to modify your code to deal with unexpected check box states. If your application calls for the use of grayed check boxes, they are also available to you.

Note Files covering all the concepts in this exercise, including any code you need to type, can be found in the Visual Basic project EX11.VBP, located under QUICK TOUR 1 in the SAMPLES subdirectory of your installation directory.

1. Create a new project and place an SSCheck control on Form1.
2. Check the value of the **TripleState** property. This is the property that determines whether the control will support two or three states. TripleState defaults to False, which maintains backwards compatibility with previous versions of THREED.
3. Place an SSCommand button on Form1 and enter the following code in the **Click** event:

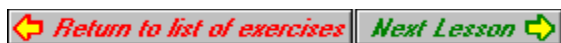
```
SSCheck1.Value = ssCBChecked
```

4. Run the project. When you click the button, the check box becomes checked. Because the grayed state is unavailable, any non-zero value evaluates as logical true, causing the control to become checked.
5. Stop the project. Select the SSCheck control and change the **TripleState** property to True. Now run the project again and click the command button. This time, the check box appears in the grayed state.

Now you are ready to use the SSCheck control in whichever mode suits the needs of your application.

This completes the SSCheck/SSOption portion of the Quick Tour. You should now have an understanding of the properties that are unique to those controls, and some of the possibilities they offer you.

The final lesson of Quick Tour 1 covers the THREED20 container controls; the SSFrame and the SSPanel. You will see that these two controls give you yet another attractive way to liven up your applications.



Exercise 12: Understanding Background Pictures

Visually, one of the coolest features of the World Wide Web is the use of tiled backgrounds. Backgrounds provide a setting and an atmosphere to the information, and create a distinctive look or mood that helps distinguish the web site from all others. ActiveThread gives you the ability to create that same sense of identity in your applications, using the SSFrame and SSPanel controls.

In addition to the control's picture, which can be animated and is primarily associated with the caption of the control, SSFrame and SSPanel support background pictures. The background picture appears behind the control, which means it is behind any child controls contained by the SSFrame or SSPanel, and behind the caption and picture of the control itself.

Background pictures can be specified in a variety of styles, including tiled, to give your application the custom look that best suits its function and your taste.

Note Files covering all the concepts in this exercise, including any code you need to type, can be found in the Visual Basic project EX12.VBP, located under QUICK TOUR 1 in the SAMPLES subdirectory of your installation directory. The indicated picture files may also be found in this directory.


1. Start a new project, and place an SSFrame control on Form1. Set the **Picture** property of Form1 to BACKGR1.BMP.
2. Set the **Picture** property of the Frame to CUBE1.BMP. Notice that the picture appears in the caption area of the control. Keep in mind that when dealing with the frame, the default **Picture** property always refers to the caption picture. In the SSPanel, the picture specified by the **Picture** property can be moved to any part of the control (as you will see in the next exercise) but is still independent of the background. You can apply all the common features to the picture specified by **Picture**; animation, color masking, etc.
3. Like the other THREED20 controls, the type of background the Frame has is controlled by the **BackStyle** property. The **BackStyle** property should be set to its default setting of '0 - Opaque.'
4. In addition to the styles specified by **BackStyle**, the SSFrame and SSPanel have a number of new settings that were not present in the other controls. These are available using the **PictureBackgroundStyle** property. The settings for this property are Picture Centered, Picture Stretched and Picture Tiled. These additional settings give you control over the display of the background picture. Make sure the **PictureBackgroundStyle** property is set to '0 - Centered.'
5. Select the **PictureBackground** property and set it to 3DTILE1.BMP. The picture appears in the center of the frame, surrounded by an area of the frame's **BackColor**. (If you do not see this, you may need to increase the size of your SSFrame.)
6. Click on the Form, and set its **ClipControls** property to False. Now select SSFrame1 and change the **BackStyle** to '1 - Transparent.' Click OK to acknowledge the transparency warning. The frame should now have a picture in the center, and be surrounded by a transparent background.

Note As with other controls, if you wish to change the transparency of an SSFrame or SSPanel at run time, the control must be set to a transparent style at design time. If you attempt to change an opaque control to transparent at run-time, an error will result.

7. Set the **PictureBackgroundStyle** of the SSFrame to '2 - Tiled.' The graphic will be repeated multiple times to fill the area of the control. You can also try the setting of '1 - Stretched' to see its effect on the control.
8. You will be using this project in the next exercise, so you may want to save it now.

Now you've seen the difference between the standard **Picture** property and the **PictureBackground** property of the SSFrame and SSPanel controls. You've also explored the different **PictureBackgroundStyle** settings of the controls that let you specify how the background will be displayed.

In the next exercise, you will see some of the unique alignment properties of the SSFrame and SSPanel controls, and learn more about using pictures with these controls.

Next Exercise 

Exercise 13: More On Alignment

Because each of the THREEED20 controls are different, they handle graphics and the alignment of pictures in slightly different ways. For example, you saw in Exercise 9 that the **PictureAlignment** options of the SSCheck and SSOption controls are left of the caption, right of the caption and the edge of the control opposite the check box/option button. These options are sufficient for Check and Option controls, but not for a control such as the Panel or the Command button, where more possibilities are needed.

Similarly, the way alignment properties are implemented varies from control to control. This exercise will show you what the differences are and how they apply to the Frame, Panel and other THREEED20 controls.

Note Files covering all the concepts in this exercise, including any code you need to type, can be found in the Visual Basic project EX13.VBP, located under QUICK TOUR 1 in the SAMPLES subdirectory of your installation directory. The indicated picture file may also be found in this directory.

1. If you do not have the project from the previous exercise available, open it now, or re-create it by following the instructions in the previous exercise. Alternately, you can use the included example.
2. Once you have a form with an SSFrame control on it, add an SSPanel control.
3. Set the **Picture** property of the SSPanel control to CUBE1.BMP. Notice the difference between the alignment of the picture on the SSFrame control and the picture on the SSPanel control. While the Frame's picture is tied to the caption area of the control, the Panel's picture is aligned relative to the area of the entire control.
4. Select the **PictureAlignment** property of the SSPanel and look at the possible values for the property. The SSPanel has the greatest number of **PictureAlignment** options of any ActiveThreed control, although the SSCommand and SSRibbon buttons have nearly as many settings as the SSPanel. Try setting **PictureAlignment** of the SSPanel to '4 - Right Middle', then to '8 - Center Bottom', then to '10 - Right of Caption', checking the effect each time. Try as many settings you'd like until you are familiar with this property.
5. Change the **PictureAlignment** property of the SSFrame control to '1 - Right of Caption.' There are only two options for aligning the picture in the Frame control.
6. You have control over more than just the alignment of the pictures used in your controls. You can also control the alignment of the caption area of the control. The settings for the **Alignment** property of each control are different, and affect the control's appearance in different ways. For example, select the SSPanel control and set its **Alignment** property to '1 - Left Justify - BOTTOM.' Notice how the caption moves to the lower left corner of the control. The picture does not move.
7. Now select the SSFrame control and change the **Alignment** property to '3 - Bottom Left'. This makes a much more noticeable change in the control. The entire caption area is moved to the bottom of the frame, and the picture moves as well. However, the alignment of the picture is unchanged with respect to the caption.

Now that you have explored some of the different caption alignment and picture alignment options of the SSFrame and SSPanel controls, you may want to go back and see what options are available for the other THREEED20 controls. You should have a better idea now

of how to change pictures and caption text independently of one another, and align them to different areas of the control.

The next lesson covers the use of the SSPanel control as a progress indicator.

Next Exercise 

Exercise 14: Making A Progress Indicator

Often, you want to display to the user a visual indicator of the progress of some task. This might be a lengthy operation, such as a file download or a lengthy database update, or simply something that happens sequentially, such as a product installation or the completion of a series of wizard-style dialogs. Whatever the reason, the SSPanel is capable of handling this type of job, and even has some unique properties to make your work easier.

1. Create a new project, and add an SSPanel control to Form1.
2. Add a horizontal scroll bar below the panel. The scroll bar will represent the "process" being measured by the panel. Set the following properties for the scroll bar:
LargeChange = 10
Max = 100
3. To set up the Panel for use as a progress indicator, you use properties, which control the manner in which the Panel is filled or "flooded" with another color. To activate the indicator features, all you need to do is set the FloodType property to a value other than 0. First, set **FloodType** to '1 - Left to Right.' This is the most common type of flood fill, but you have several other choices, including vertical fills.
4. You may notice that the caption of the control is no longer visible. This is because when using a flood fill style, the caption is disabled. However, you can have the control display the percentage of completion in place of the caption. Set the **FloodShowPct** property to True.
5. To set the amount of completion, you use the run-time only **FloodPercent** property. Double-click the scroll bar and enter the following code in the **Change** event:

```
SSPanel1.FloodPercent = Hscroll11.Value
```

6. Run the project. Click on the right button of the scroll bar. As the scroll bar's thumb moves, you will see the progress indicator fill up with blue, and the percentage of completion will be displayed. When the scroll bar thumb is all the way to the right, the SSPanel reads 100%. Stop the project.
7. You can choose from a variety of progress indicator effects. Select the SSPanel and change the following properties:
FloodColor = &H00000080& (Dark red)
FloodFillStyle = '1 - Segmented'
8. Run the project again and drag the scroll bar. This time the indicator fills up with dark red in discrete segments. Stop the project.
9. There is one final setting you can explore. Resize the SSPanel so that it's shape is mostly square, but fairly large. Move or resize the scroll bar as necessary. Finally, set **FloodType** to '4 - Widening Circle.' Run the project. This time, the indicator takes the form of a colored circle that grows. When the edges of the circle touch the edge of the control, the task is complete.

Congratulations! You have now completed all of Quick Tour 1. You have gained a great deal of knowledge that will help you to use the ActiveThread THREED20 controls to their maximum potential. Starting with the common properties, you have covered all the major

features of each THREEED20 control. You now know the majority of what you need to know to use the controls to activate your programs with multimedia power. A few topics, such as data binding, were not covered by this tutorial, but you can find out what you need to know about these topics by consulting the Control Reference. For questions on individual properties, methods or events, click the appropriate heading on the contents page and choose the item of interest off the list.


Quick Tour 2 introduces you to the concepts and usage of the SSSplitter control. The Splitter is a flexible container control that allows you to organize multiple controls in a series of resizable panes. When you drag the splitter bars that separate these panes with your mouse, the control contained by each pane is automatically resized. The SSSplitter controls brings the frames metaphor, found on many web sites, to your desktop applications.

 [*Return to list of exercises*](#)

Exercise 1: Adding A Scrolling Caption


In this exercise, you will learn how to create a scrolling, or *marquee* caption. Marquee captions use motion to draw the user's attention to a particularly important feature of your program. For maximum effectiveness, marquee captions should be used sparingly throughout your interface.

Note Files covering all the concepts in this exercise, including any code you need to type, can be found in the Visual Basic project EX1.VBP, located under QUICK TOUR 1 in the SAMPLES subdirectory of your installation directory.

1. Start Visual Basic by clicking on the  icon on the Start Menu or Program Manager.
2. If the ActiveThread THREED20 controls do not appear in your toolbox, open the Custom Controls dialog. When the list of available controls appears, select "Sheridan ActiveThread Controls" and click OK.
3. Begin by placing an SSCommand button on the form. Select the



button from the Toolbox and draw a fair sized button on the form. Although you will be working with an SSCommand button for the first part of this lesson, keep in mind that the properties used may be applied to any THREED20 control.

4. Change the **Caption** property to a long string of text, such as "This is the Introduction to Sheridan ActiveThread Controls". A long text string will better illustrate some of the changes you will be making to the control.
5. The caption text appears centered on the button. Depending on the size of your command button, the beginning and end of the long caption may be cropped by the edges of the control. This is the default behavior for caption text.
6. Now change the setting of the **CaptionStyle** property to '1 - Wrapped.' Notice how the text wraps to multiple lines so that all the text fits within the control. You may need to adjust the height of your control to see this effect. Change **CaptionStyle** back to '0 - Standard.'
7. To get the caption to scroll, you must set the **MarqueeStyle** property. The default setting is '0 - None' which results in a static caption. Change **MarqueeStyle** to '1 - Scrolling.'
8. Run the project. (You do not need to save it at this point.) You will see the caption scrolling slowly from right to left. Take note of the speed and smoothness of the scrolling motion. Also notice that even if the caption text is longer than the width of the button, you can read the entire caption as it scrolls. Close the form or press  to stop the program.
9. The speed and smoothness of the scrolling are controlled by the **MarqueeDelay** and **MarqueeScrollAmount** properties. To see the effects of these properties, first change **MarqueeScrollAmount** from 1 to 20. Leave **MarqueeDelay** the same for now. Run the project again. You will see the caption scrolling much more quickly, but you may notice that the scrolling motion is rather choppy. **MarqueeScrollAmount** sets the number of pixels the caption will move during each "scroll". Increasing its value causes the caption to move more quickly, at the expense of smoothness.

10. Stop the program and change the setting of **MarqueeScrollAmount** back to 1. Change the value of **MarqueeDelay** to 0. Run the project again and observe the difference. The scrolling is still faster than it was originally, but much smoother. **MarqueeDelay** controls the number of milliseconds between each "Scroll." By reducing its value you can increase the speed of the scrolling without sacrificing smoothness. You should be careful, however, because too low a setting for **MarqueeDelay** makes the control consume more CPU resources, affecting overall system responsiveness. Stop the program and reset **MarqueeDelay** to 120.
11. Now that you know how to control the scrolling motion of the text, try changing the direction of the scrolling. You have seen the text scrolling horizontally from right to left. As you might expect, you can also reverse direction and have the text scroll from left to right. To make things interesting though, select the **MarqueeDirection** property and set it to '3 - Bottom to Top.' Before you run the project, notice that the caption has changed and now appears wrapped, as it did in step 6. This is because the caption text wraps automatically for vertical scrolling. Similarly, if you had left **CaptionStyle** set to '1 - Wrapped' in step 6, you would have seen the text change to a single line when you initially selected the scrolling **MarqueeStyle**. Now run the project. The text scrolls vertically in a style reminiscent of movie credits. Stop the project when you have seen enough.
12. Scrolling is only one type of movement that is available to caption text. Change the height of your command button so that the caption fits completely within the control, with room to spare both below and above it. Now change the **MarqueeStyle** property to '4- Bouncing.' Run the project again. You will see the caption text scroll to the top of the control as it did before, but when it reaches the top, it reverses direction and moves the other way. Once you have seen how this works, stop the project.

You can explore the remaining **MarqueeStyle** settings, Blinking and Sliding, on your own. Blinking flashes the caption on and off at a rate specified by **MarqueeDelay**, while Sliding is similar to Scrolling, except that the text stops moving once all of it has been displayed.
13. Now, double click your command button to bring up the code window and select the **MarqueeCycleBegin** event. This is the event that is fired whenever a marquee caption appears or changes direction. Enter the following code in the **MarqueeCycleBegin** event procedure:

```
Select Case SSCommand1.ForeColor
Case vbBlack
    SSCommand1.ForeColor = vbBlue
Case vbBlue
    SSCommand1.ForeColor = vbCyan
Case vbCyan
    SSCommand1.ForeColor = vbGreen
Case vbGreen
    SSCommand1.ForeColor = vbMagenta
Case vbMagenta
    SSCommand1.ForeColor = vbRed
Case vbRed
    SSCommand1.ForeColor = vbYellow
Case vbYellow
    SSCommand1.ForeColor = vbWhite
Case vbWhite
    SSCommand1.ForeColor = vbBlack
```

End Select

Once you have entered and checked the code, run the project. You will see the color of the text change each time the marquee caption begins a new cycle. With the present settings, a cycle begins each time the caption bumps against the edge of the control and reverses direction.

Now you know how to create marquee style captions of various types, control their layout, speed and direction, and use the **MarqueeCycleBegin** event to take action based on the position of the marquee text. Again, you can apply these properties to all the ActiveThread controls. Certain controls may have limitations or implement a particular property slightly differently than the SSCommand button, so be sure and double check the documentation for any property you need to use.

Since you now have the ability to add visual appeal to your applications with moving text, the next lesson will show you how to add audio appeal by using sounds with your ActiveThread controls

<input checked="" type="radio"/> Left Justified SSOption	Right Justified SSOption <input type="radio"/>
<input type="checkbox"/> Left Justified SSCheck	Right Justified SSCheck <input checked="" type="checkbox"/>

Exercise 2: Playing Sounds

Adding sound to your application with ActiveThreed controls is simple and straightforward. Sounds enhance the user interface of your application, and give it a true multimedia look and feel.

Note Files covering all the concepts in this exercise, including any code you need to type, can be found in the Visual Basic project EX2.VBP, located under QUICK TOUR 1 in the SAMPLES subdirectory of your installation directory. The indicated sound files may also be found in this directory.

1. Create a new project, and place an SSCommand button on Form1.
2. Double-click the command button to bring up the code window. Add the following statement to the **Click** event:

```
SSCommand1.PlaySoundFile "3DCLICK.WAV"
```

The sound file is supplied with ActiveThreed, but you could use one of your own. Whichever you choose, the sound clip should be fairly short.

Note In the design environment, the .WAV file must be located in the same directory as your Visual Basic executable file, or in one of the directories referenced by the environment path. Alternatively, you can specify the full path information for the file in the **PlaySoundFile** method. When you distribute applications, the sound file may be stored in the same directory as your executable file.

3. Run the project. When you click the button, you should hear the sound effect.
4. Stop the program. Now place an SSPanel control on the form and set the following properties for it:
MarqueeStyle = 'Bouncing'
MarqueeScrollAmount = 5
5. Next you will specify a sound file that will provide an audio cue whenever the caption bounces against the edge of the control. The included 3DBINK.WAV file is suitable for this task.

The place to play the sound is the **MarqueeCycleEnd** event. This is the event that is fired when the marquee caption changes direction, so it is the best place to put a sound effect that corresponds to that occurrence.

Add this code to the **MarqueeCycleEnd** event of the SSPanel control:

```
SSPanel1.PlaySoundFile "3DBINK.WAV"
```

6. Run the project. As the caption on the panel scrolls, you will hear a sound effect whenever it bumps against the edge of the panel.

As you can see, adding sound to the ActiveThreed controls is easy. In the next exercise, you will learn how to add a cool, Web-like interface to your applications by using transparent controls.

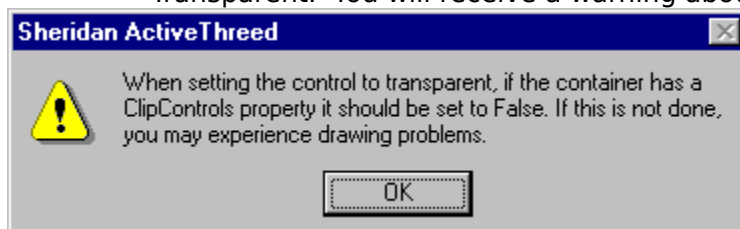
<input checked="" type="radio"/> Left Justified SSOption	Right Justified SSOption <input type="radio"/>
<input type="checkbox"/> Left Justified SSCheck	Right Justified SSCheck <input checked="" type="checkbox"/>

Exercise 3: Transparent Operation

The success of the World Wide Web, coupled with the evolution of computer display technology, has caused a shift in the nature of user interface design. The metallic elements and single color backgrounds found in many programs have begun to give way to colorful images and textures. Using ActiveThreed controls, you can present a Web-like interface to your users while using familiar component technology. One of the features that helps you to do this is the transparent background option.

Note Files covering all the concepts in this exercise, including any code you need to type, can be found in the Visual Basic project EX3.VBP, located under QUICK TOUR 1 in the SAMPLES subdirectory of your installation directory. The indicated picture file may also be found in this directory.

1. Start a new project, and set the **ClipControls** property of Form1 to False. In order for transparent backgrounds to function properly, the **ClipControls** property of the control's container must always be set to False.
2. Select a background picture for your form. This can be any Windows bitmap that is large enough to cover the form, or you can use the included BACKGR1.BMP. Specify the file name of your bitmap in the **Picture** property of Form 1.
3. Place an SSFrame control on the form. Set the **BackStyle** property to '1 - Transparent.' You will receive a warning about setting **ClipControls** to False.



This warning appears the first time you change a control's **BackStyle** to transparent. Since you have already set the form's **ClipControls** property to False, simply click OK to acknowledge the warning.

4. Without even running the project, you can see that the Frame has become transparent.
5. Now add an SSCommand button to the form beside the SSFrame. Again, set the **BackStyle** to '1 - Transparent.'
6. Next, you will add code to the command button to change the transparency when the mouse passes over it. Double-click the SSCommand button to open its code window.

Note A control that has a transparent background specified at design time can easily be changed to another type of background at run time. *The reverse is not true.* If a control is given an opaque **BackStyle** at design time and you attempt to change it to a transparent **BackStyle** at run time, an error will result. If you wish to change the transparency of a control through code, it must first be set to transparent at design time.

7. Select the **MouseEnter** event for the command button, and place the following code in the event:

```
SSCommand1.BackStyle = ssOpaque
```

8. Select the **MouseExit** event and place the following code in the event procedure:

```
SSCommand1.BackStyle = ssTransparent
```

9. Run the project, and pass the mouse pointer over the command button. The button now becomes opaque whenever the mouse is over it. Stop the project when you are done.

Now you have seen how transparency works, and begun to explore the use of mouse events to control the interface. The use of transparency will be more fully explored in the lessons that follow.

<input checked="" type="radio"/> Left Justified SSOption	Right Justified SSOption <input type="radio"/>
<input type="checkbox"/> Left Justified SSCheck	Right Justified SSCheck <input checked="" type="checkbox"/>

Exercise 4: Adding an Animated Picture

Any of the ActiveThreed controls can have an animated picture in addition to or in place of its caption. Like marquee captions, animation adds visual pizzazz to your application and draws attention to critical features. As with animated captions, animated pictures are best used sparingly to ensure maximum effect.

Note Files covering all the concepts in this exercise, including any code you need to type, can be found in the Visual Basic project EX4.VBP, located under QUICK TOUR 1 in the SAMPLES subdirectory of your installation directory. The indicated picture file may also be found in this directory.

1. Create a new project and place a large SSCommand button on Form1. To make the picture easier to see, delete the text of the **Caption** property.
2. Animated pictures use the standard **Picture** property of the control. You supply a special segmented bitmap that has been divided into a number of areas of equal size. The segments of the bitmap become the frames of your animation. The Property Pages for the ActiveThreed controls contain a special utility to help you construct a segmented bitmap out of a series of individual bitmaps.

For now, you can use a picture file that is included with ActiveThreed. Specify ANIM19.BMP for the command button's **Picture** property.

3. Once you have specified a segmented bitmap for the **Picture** property, you must tell the control how many segments the bitmap contains. This is done using the **PictureFrames** property. By default, **PictureFrames** is set to 1 for a static image. The ANIM19.BMP image contains 19 segments, so set **PictureFrames** to 19. The control should look like this:



4. Run the project and watch the show! Creating an animated control is that simple. ActiveThreed also gives you a finer degree of control over the animation. Stop the project and return to design mode.
5. As with the marquee caption properties, you can control the delay and therefore the speed of the animation. Set the **PictureAnimationDelay** property to 200 and run the project again. You will see the animation occurring much more slowly.
6. Not only can you control the speed of the animation, you can stop it altogether. Double-click on the SSCommand button and enter the following code in the **Click** event:

```
SSCommand1.PictureAnimationEnabled = Not  
SSCommand1.PictureAnimationEnabled
```

Run the project and click the button a few times to see the effect. The **PictureAnimationEnabled** property simply turns the animation on and off at the current frame.

7. You will be using this project in the next exercise, so you may want to save it now.

There are other properties that can have a profound effect on the pictures and animations you use with your controls. These properties give you the ability to make parts of the graphic transparent, and automatically adjust your controls to fit the size of any picture or animation. These properties are covered in the next exercise.

<input checked="" type="radio"/> Left Justified SSOption	Right Justified SSOption <input type="radio"/>
<input type="checkbox"/> Left Justified SSCheck	Right Justified SSCheck <input checked="" type="checkbox"/>

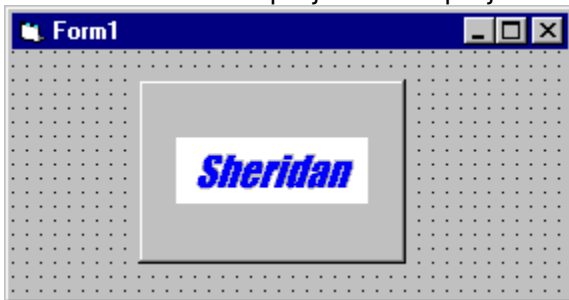
Exercise 5: Mask Colors & AutoSize


Whenever possible, you want the graphics used in your application to blend naturally with the rest of the interface. Many applications sport custom textured backgrounds, and most users of the 32-bit versions of Windows can alter their background colors to suit personal taste. One way to insure a sophisticated looking interface is to use transparent areas in your graphics. Transparency allows background colors and textures to show through, and lets your graphics appear to be non-rectangular. ActiveThreed gives you the ability to create transparent graphics using *mask colors*.

In addition to using mask colors, you can have the ActiveThreed controls adjust the size of the picture to match the control, or resize the control to the size of the picture. These techniques are useful when you want to altogether replace the face of a button, panel or other control with a graphic.

Note Files covering all the concepts in this exercise, including any code you need to type, can be found in the Visual Basic project EX5.VBP, located under QUICK TOUR 1 in the SAMPLES subdirectory of your installation directory. The indicated picture file may also be found in this directory.

1. If you do not have the project from the previous lesson open, open it now. If you must, re-create it by following the steps in Exercise 4. Alternately, you can use the included project. Your project should look like this:



2. In order to use a mask color, you must specify which color will be masked out of the image. To make the white area around the animation transparent, first set the **PictureMaskColor** property to white. Click on the  button for the **PictureMaskColor** property in the Property window, and choose white from the color palette. Alternatively, you can enter &H00FFFFFF& for the property's value.
3. Once you have selected the color which will be masked, you must turn on masking to enable the transparent effect. Do this by setting **PictureUseMask** to True. You should immediately see the results of the color masking. The project will now look like this:



4. After setting the transparency, you may want to adjust the size of the control to conform to the size of the picture, or you might want the picture scaled to fill the control at its present size. In either case, you can use the **AutoSize** property to

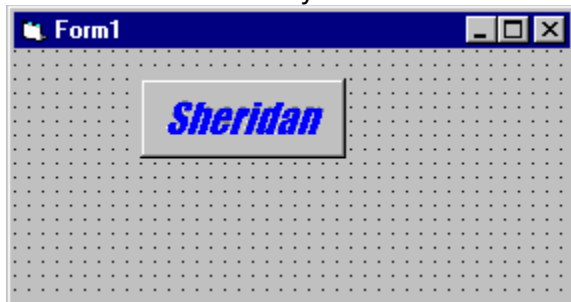
adjust the dimensions of the graphic and the control relative to one another. This will be especially useful later when working with button styles.

Note The **AutoSize** property only functions when the control has no caption text. If you have caption text specified for your control, delete it now.

Set the **AutoSize** property of your command button to '1 - Adjust Picture Size to Button.' Your project should now look like this:



5. Change the setting of **AutoSize** to '2 - Adjust Button Size to Picture.' You will immediately see the difference:



6. You will be using this project in the next exercise, so if you are not using the sample projects included with ActiveThread, save the project now.

Now you have seen how to make your control graphics transparent and automatically adjust the size of the controls to match the graphic. Although you have worked primarily with **SSCommand** buttons, the skills you have learned are applicable to any of the **THREED20** controls.

In the next exercise, you will learn how to further expand upon the concepts of the last two lessons and create even more customized interface effects. The properties covered in the next few exercises will apply only to buttons - the **SSCommand** and **SSRibbon** controls.



Exercise 6: Border Styles & Button Pictures

This exercise explores some of the properties that are unique to the ActiveThread button controls - SSCommand and SSRibbon. You can use them to create a customized interface style of your choice, from borderless, web-like buttons to Office-style toolbars.

Note Files covering all the concepts in this exercise, including any code you need to type, can be found in the Visual Basic project EX6.VBP, located under QUICK TOUR 1 in the SAMPLES subdirectory of your installation directory. The indicated picture file may also be found in this directory.

1. If you do not have the project from the last exercise available, open it now. If you did not save the project from the last exercise, you may use the one provided.
2. The SSCommand and SSRibbon buttons support a variety of border styles to match the chosen look of your application. By default, the appearance of the button follows that of the operating system's native controls. You can change this by changing the setting of the **ButtonStyle** property. Try changing **ButtonStyle** from the default value ('0 - Follow OS') to '3 - Active Borders'. Your form should look like this:



Run the project. The button appears to have no border, but when the mouse pointer passes over the control, the border becomes visible.

Note that the control will display a focus rectangle if it is the only control on the form, regardless of its current border state. This is normal behavior for a command button. You can add a second control, such as a text box, and give that control focus if you want to see what the button looks like without the focus rectangle.

If you do not want a focus rectangle to appear, you can use an SSRibbon button, which cannot receive input focus and therefore will not display a focus rectangle. The down side to this approach is that the user must click an SSRibbon button with the mouse. The spacebar cannot be used to click an SSRibbon because the control cannot receive focus. Also, a Ribbon button operates in toggle mode, so you must add code if you wish to duplicate the functions of a Command button.

3. You can use a combination of properties to create buttons with a completely custom appearance. For example, you can create a button that appears to be round instead of square. The next few steps will show you how to do this. First, add a second SSCommand button to your form and set the properties as follows:
AutoSize = '2 - Adjust Button Size to Picture'
BackStyle = '1 - Transparent'
ButtonStyle = '4 - Borderless'
Caption = "" (empty)

4. Make sure the form has the following properties set to the following values:
ClipControls = False
Picture = BACKGR1.BMP (in the QUICK TOUR 1 subdirectory)
5. Set the **Picture** property of SSCommand2 to REDBALL1.BMP. This file can be found in the QUICK TOUR 1 subdirectory of the SAMPLES directory.
6. Change the **PictureMaskColor** to white. You can select it from the property sheet color palette, or enter the value &H00FFFFFF& for the property. Make sure **PictureUseMask** is set to True.
7. Run the project. You will see that your second SSCommand button appears to be a red ball floating over the textured background. Only the focus rectangle is rectangular. If you click on the first button, the focus rectangle will move, and the button will appear perfectly round.

Now that you know how to design a custom button appearance, you are ready to go a step further and define different appearances for the different states of the control. The next exercise will build on the form and controls created in this exercise, so you may want to save your project now.

<input checked="" type="radio"/> Left Justified SSOption	<input type="radio"/> Right Justified SSOption
<input type="checkbox"/> Left Justified SSCheck	<input checked="" type="checkbox"/> Right Justified SSCheck

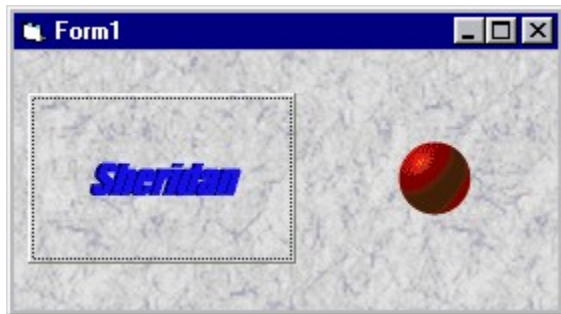
Exercise 7: Adding Additional Button States

This exercise will show you how to control pictures and animation for the various states of the command and ribbon buttons. By using button state pictures, you gain complete control over the appearance of the button in all phases of its operation.

All buttons have at least three states, and *up* state, which is the button's normal appearance, a *down* state, which is how the button appears when depressed, and a *disabled* state, which is how the button appears when it is inactive. The SSRibbon button adds a fourth state, *down disabled*, to cover inactive, depressed buttons. You can specify a separate picture (or animation) for each of these states. If you wish your button to appear transparent, each state picture must share a common mask color.

Note Files covering all the concepts in this exercise, including any code you need to type, can be found in the Visual Basic project EX7.VBP, located under QUICK TOUR 1 in the SAMPLES subdirectory of your installation directory. The indicated picture files may also be found in this directory.

1. If you do not have the project from the last exercise available, open it now. If you did not save the project from the last exercise, you may use the one provided. Your project should have two SSCommand buttons; SSCommand1 is animated with active borders, SSCommand2 is a borderless red ball. Both buttons should use mask colors and have transparent backgrounds.



2. Select SSCommand2 (the red ball) and choose the **PictureDn** property from the property sheet. In the FileOpen dialog, select the file GRNBALL1.BMP and click OK. Make sure the **PictureDnFrames** property is set to 1, since this button is not animated.
3. Run the project and click the red ball. When you press it with the mouse, the ball turns green! Stop the project.
4. Select the red ball button again and choose the **PictureDisabled** property from the property sheet. In the File Open dialog, select GRYBALL1.BMP and click OK. Make sure the **PictureDisabledFrames** property is set to 1, since this button is not animated.
5. Select SSCommand1 (the animated one) and add the following code to the **Click** event:

```
SSCommand2.Enabled = Not SSCommand2.Enabled
```
6. Run the project. First, click the red ball to verify that it turns green when clicked. Then click the animated button. You should see the red ball turn gray as the control is disabled. Verify this by attempting to click on the gray ball. When you are done, click the animated button again to return the ball to its enabled state. Then stop the project.

7. Now that you have seen how to specify the down and disabled states of a static button, changing the states of an animated button should be easy! Select `SSCommand1`, and from the property sheet choose **PictureDisabled**. In the File Open dialog, select the file `DIANIM19.BMP` and click OK.
8. Once you have selected the disabled state picture, change **PictureDisabledFrames** to 19, the number of frames in this animation.
9. Enter the following code in the **Click** event of `SSCommand2`:

```
SSCommand1.Enabled = Not SSCommand1.Enabled
```

10. Run the project. This time click on the red ball first. You will see that the animation turns gray, indicating the button is disabled. Because both the enabled and disabled animations have the same number of frames, the animation continues smoothly between the two states. If the animations had different numbers of frames, changing the state would start each animation at the beginning.

If you pass the mouse over the animated button, you may also notice that the borders do not "activate," another indication of the button's inactive state. Click the red ball again to re-enable the animated button. Try clicking it now. When you are done, stop the project.

11. To see the effects of the fourth state, you will need an `SSRibbon` button. Create an `SSRibbon` button on your form and give it the following property settings:

AutoSize	= '2 - Adjust Button Size to Picture'
BackStyle	= '1 - Transparent'
ButtonStyle	= '4 - Borderless'
Caption	= "" (empty)
Picture	= <code>BLUBALL1.BMP</code>
PictureDisabled	= <code>GRYBALL1.BMP</code>
PictureDisabledFrames	= 1
PictureDn	= <code>BLUBALL2.BMP</code>
PictureDnDisabled	= <code>GRYBALL2.BMP</code>
PictureDnDisabledFrames	= 1
PictureDnFrames	= 1
PictureFrames	= 1
PictureMaskColor	= <code>&H00FFFFFF&</code> (white)
PictureUseMask	= True

12. After you have created the ribbon button, add the following code to the **Click** event of `SSCommand1`:

```
SSRibbon1.Enabled = Not SSRibbon1.Enabled
```

13. Run the project. Click the ribbon button a few times. Because the `SSRibbon` button is a toggle button, you will see the control switch between the up and down state once with each click. Click the ribbon button to return it to the up state.
14. Click `SSCommand1` to disable the ribbon button. Notice that in the up state, the ribbon button looks the same as `SSCommand2` when disabled. Click `SSCommand1` again to re-enable both buttons.
15. Click the ribbon button to toggle it into the down state. Now click `SSCommand1` again to disable it. This time you will see the fourth state of the button come into

effect. When you have seen enough, stop the project.

This exercise has covered the concept of button states in depth, shown you how to set different appearances and even animations for the various button states. It has also served as an introduction to the special features found in ribbon buttons, and the differences between ribbon and command buttons.

The next exercise will be dedicated to the SSRibbon button and the capabilities of that control. You will see how to use ribbon buttons in groups to create exclusive and toggle button effects.

Left Justified SSOption Right Justified SSOption
 Left Justified SSCheck Right Justified SSCheck

Exercise 8: Using Buttons In Groups

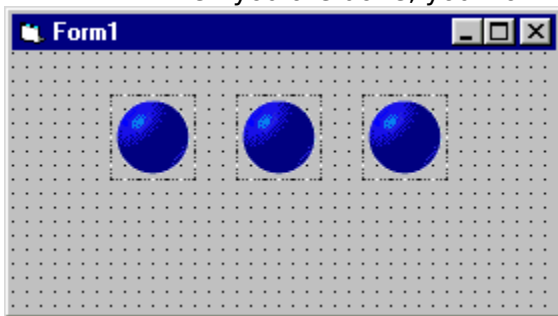
By this time, you know most of what you need to know about the common properties of the THREEED20 controls. You have also learned about some of the unique features of the SSCommand and SSRibbon controls. In the final exercise of this lesson, you will discover some of the features of the SSRibbon button that make it unique. Most of these features involve using a series of SSRibbon buttons together, as you might on a toolbar, or if you wished to emulate an option button using an SSRibbon button.

Note Files covering all the concepts in this exercise, including any code you need to type, can be found in the Visual Basic project EX8.VBP, located under QUICK TOUR 1 in the SAMPLES subdirectory of your installation directory. The indicated picture files may also be found in this directory.

1. Create a new project and place three SSRibbon buttons on Form1.
2. Make sure the **ClipControls** property of the form is set to False, then set the following properties for each button (You may want to create a single button, set the properties, then copy the button and paste it back onto the form twice. Although you could create a control array, for the purposes of this exercise *do not* create a control array.):

AutoSize	= '2 - Adjust Button Size to Picture'
BackStyle	= '1 - Transparent'
ButtonStyle	= '4 - Borderless'
Caption	= "" (empty)
Picture	= BLUBALL1.BMP
PictureDn	= BLUBALL2.BMP
PictureDnFrames	= 1
PictureFrames	= 1
PictureMaskColor	= &H00FFFFFF& (white)
PictureUseMask	= True

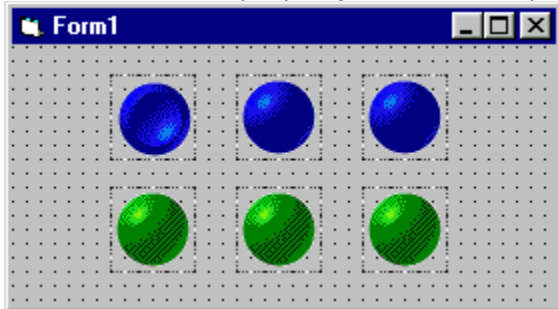
When you are done, your form should look something like this:



2. Click on each button, one at a time, and check the value of the **GroupNumber** property. Each button should be set to group number 1, as this is the default. This means the buttons will interact with one another in an exclusive fashion. Also check the value of the **GroupAllowAllUp** property. This should be set to True, meaning that it is possible for all the buttons in the group to be in the up position (when a button is in the up position, its **Value** = False.)
3. Run the project. Click on the first button. It will remain pressed. Click the first button a second time, and it will return to the up position.

4. Click the first button a third time so it is pressed. Now click on the second button. As the second button is pressed, the first button returns to the up position. Click the third button and the second button becomes unpressed. Finally, click the third button again. Now all buttons should be in the up position.
5. Stop the project. On the form, select the first button and change the **GroupAllowAllUp** property to False. Now select the third button and check the value of the **GroupAllowAllUp** property. You will see that it has also changed to False. Setting the **GroupAllowAllUp** value of one SSRibbon button sets it for all the buttons in the same group.
6. Run the project again. Notice that although you have set **GroupAllowAllUp** to False, all the buttons in the group are up! This is because changing the value of **GroupAllowAllUp** will not change the **Value** property of any individual button. **GroupAllowAllUp** cannot take effect until you change the **Value** of a particular button. This is true both at design and run time.
7. Click the first button, then click it again. Instead of returning to the up position, the button stays pressed. Now the only way to return a button to the up position is to press another button. Once you understand how **GroupAllowAllUp** works, stop the project.
8. Select the three buttons on your form using the mouse, then copy them to the clipboard. Paste them back onto the form, answering No when asked if you want to create a control array. Move the three new buttons below the original three, and change the following properties for each one:

Picture	= GRNBALL1.BMP
PictureDn	= GRNBALL2.BMP
GroupNumber	= 2
GroupAllowAllUp	= True
9. Select the first button of the original group (the first blue button) and change its **Value** property to True. The project should now look like this:



10. Run the project. Click on several buttons from each group. You will see that the two groups are independent of one another. Clicking one of the green buttons has no effect on any of the blue buttons. Also, one of the blue buttons is always down, while all the green buttons can be up.
11. Stop the project. This time, select each button one at a time, set its **GroupAllowAllUp** property to True, and set its **GroupNumber** to a unique value (1 for SSRibbon1, 2 for SSRibbon2, etc.) Run the project again and experiment with clicking all the buttons.

Although the buttons *appear* to be grouped because of their color and arrangement, each button in this case functions independently of the others. This is an important point to consider - you must judge whether to put buttons in groups based solely on the need for exclusive operation, not on similarity of

function or appearance. What the user ultimately perceives as a "group" of ribbon buttons is not necessarily a group from the programmer's point of view.

This concludes Lesson Two. You have received a complete overview of the properties and events common to the THREEED20 controls. You should now be comfortable creating several types of scrolling captions, adding sounds, working with transparent controls, and using different types of graphics and animation, including transparent graphics.

You should also have a good feel for how to use the button controls, and an understanding of what the two types of buttons are and how they differ. You know most of the properties that are unique to the button controls, and you have an understanding of what button states are and how they work.

You now have everything you need to dive in and begin creating activated programs using ActiveThread. Start spicing up drab interfaces with hot multimedia ingredients that will make your users sit up and take notice! As you explore the possibilities, consult the Control Reference section of the help file if you need help using one of the common properties with a specific control.

The next two lessons will go into further control-specific detail, gradually covering all the features of the THREEED20 controls. Lesson 2 will teach you how to use the SSCheck and SSOOption controls, and Lesson 3 covers the special features of the SSFrame and SSPanel controls.

 [Return to list of exercises](#) [Next Lesson](#) 

Exercise 9: Captions, Pictures and Alignment

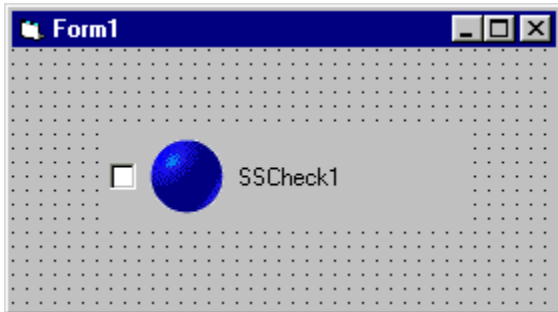
In Lesson 1, you learned about using pictures (specifically animated pictures) and captions, but not about using them together. This is because for most controls the interaction of caption and picture is straightforward; the caption appears on top of the picture, and the two have little effect on one another except for aesthetic considerations.

With the SSCheck and SSOption controls, the interaction of picture and caption is somewhat more complex. (This is also true of the SSFrame control, as you will see in a future exercise.) The area occupied by the caption is separate from the area occupied by the picture. The style and placement of the caption affects the picture, and vice-versa. This lesson will explore the nature of that interaction.

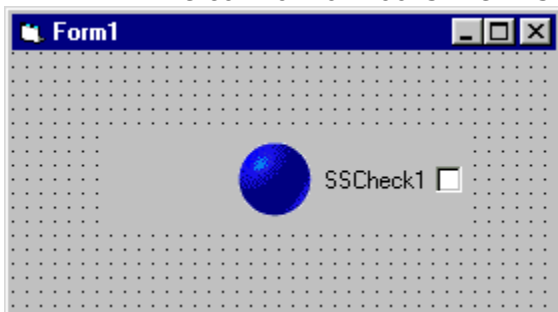
Note Files covering all the concepts in this exercise, including any code you need to type, can be found in the Visual Basic project EX9.VBP, located under QUICK TOUR 1 in the SAMPLES subdirectory of your installation directory. The indicated picture file may also be found in this directory.

1. Create a new project, and place a medium size SSCheck control on Form 1.
2. Add a picture to the control by setting the following properties:
Picture = BLUBALL1.BMP
PictureFrames = 1
PictureMaskColor = &H00FFFFFF& (white)
PictureUseMask = True

Your form should look something like this:



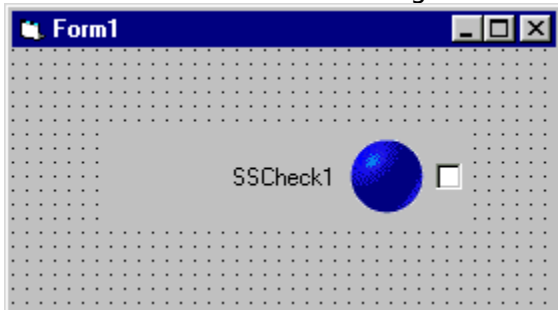
3. Notice that the picture is aligned to the left of the caption, not to the left side of the control. Also notice that the caption is not superimposed on top of the picture. In an SSCheck or SSOption, the check box or option button always stays on the outer edge of the control. Pictures or captions are aligned accordingly. You can verify this by setting the **Alignment** property of the control to '1 - Right Justify.' The control now looks like this:



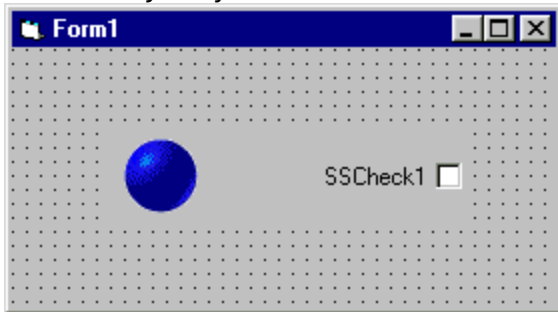
Alignment controls the orientation of the control as a whole, as determined by

the placement of the check box or option button. The picture itself may be aligned to the left or right of the caption by setting the **PictureAlignment** property.

4. Set the **PictureAlignment** property of the control to '1 - Right of Caption.' you will see the following:



There is one other setting for **PictureAlignment**. Set **PictureAlignment** to '2 - Justify.' The control assumes this appearance:



The '2 - Justify' setting aligns the picture to the edge of the control opposite the check box or option button. This is useful if you wish to have a series of controls with evenly aligned pictures.

5. There is one more consideration when using pictures with **SSCheck** and **SSOption** controls, and it involves the movement of captions using the moving **MarqueeStyle** settings. First, set the **PictureAlignment** property to '0 - Left of Caption.' Notice how the picture moves near to the end of the caption text. Then set the **MarqueeStyle** of the control to '1 - Scrolling.' The picture moves again to the edge of the control, as it was when **PictureAlignment** was set to '2 - Justify.' Although the picture's alignment hasn't changed, it has moved in response to a change in the size of the caption area. When an **SSCheck** or **SSOption** control is using a moving marquee style, the caption area of the control is maximized to better display the marquee effect.
6. Run the project. Observe the movement of the caption. Unlike the button controls, the caption does not scroll across the picture. Scrolling takes place in the caption area, which is independent of the picture area.

Now that you have covered the different alignment settings of the **SSCheck** & **SSOption** controls. In the next exercise, you will see how to customize the appearance of the interface portion of the control - that is, how to change the actual "box" portion of the check box, and the "button" of the option button.

Left Justified SSOption

Right Justified SSOption

Left Justified SSCheck

Right Justified SSCheck


Exercise 1: Setting Up Splitter Panes

Here's the first exercise in the Splitter Quick Tour. You will learn how to create a Splitter control and set up a basic system of panes.

A Splitter is a container control composed of a collection of *pane objects*. Each pane object is capable of holding a single control, although that control may be a container control. The panes of a Splitter are separated by *splitter bars*, which provide the user interface of the Splitter. Splitter bars divide two or more panes either horizontally or vertically.

You create new panes in the Splitter by splitting an existing pane into two, either horizontally or vertically. The result is that the existing pane becomes half its current width or height, and a new pane is created that is half the current pane's width or height. The two panes occupy the same area as the one pane did previously, so the size and arrangement of other existing panes is not affected. Once you see the control in operation, this procedure will be intuitively obvious.

Note Files covering all the concepts in this exercise, including any code you need to type, can be found in the Visual Basic project EX1.VBP, located under QUICK TOUR 2 in the SAMPLES subdirectory of your installation directory.


1. Start Visual Basic by clicking on the  icon on the Start Menu or Program Manager.
2. Select the SSSplitter control and create a large splitter that covers most of your form. If the ActiveThread Splitter control does not appear in your toolbox, open the Custom Controls dialog. When the list of available controls appears, check the box next to the "Sheridan Splitter Control" and click OK.
3. When the splitter appears, there is already one pane in existence. In the center of each pane is a cluster of user interface controls that looks like this:





This is the design-time interface of the SSSplitter control. These controls provide information about the current pane, and give you the ability to split or delete the pane. The individual parts of the interface are:

Pane A **Pane Name** The SSSplitter control automatically generates a default name for a pane when it is created, but you can change the name to anything you want.

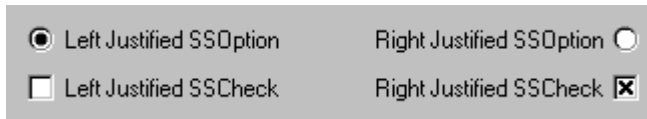
Panes(0) **Pane Number** Each pane is a member of the Panes Collection, and has a unique index. As panes are added to and deleted from the collection, the index of any given pane may change.

 **Split Horizontal** Causes the pane to be split horizontally, creating a new pane underneath the existing one.

 **Split Vertical** Causes the pane to be split vertically, creating a new pane to the right of the existing one.

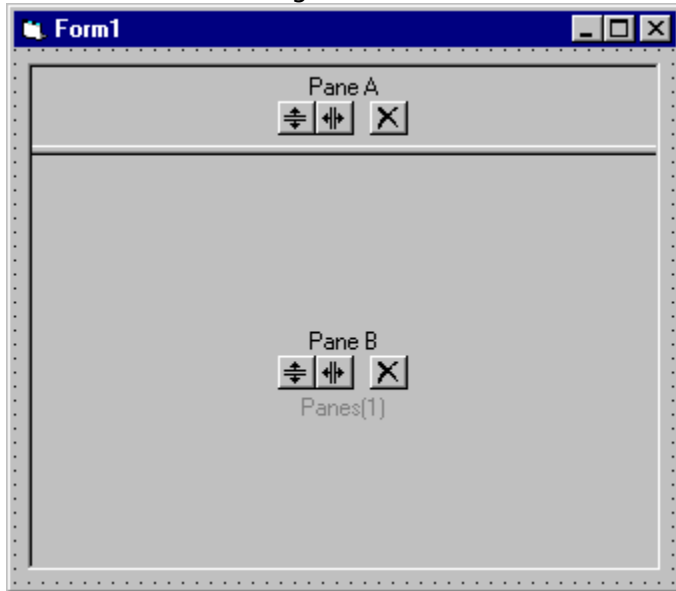
 **Delete Pane** Deletes the current pane. There must always be at least one pane object, so when the Splitter has only one pane (as it does when first created) this button will be disabled.

By using the buttons, you will see how easily and quickly you can create even complex pane layouts.



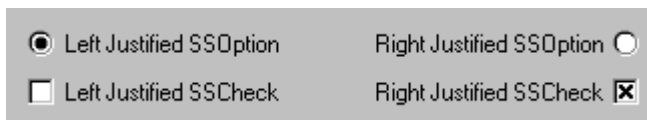
4. Click on the button of Pane A.
This will create a second pane, B, in the bottom half of the control.

5. Click on the splitter bar between Panes A and B with your mouse, and drag it towards the top of the control, until Pane A takes up only about 1/5th of the control's area. As you can see, the splitter bars operate at design time just as they do at run time. The control should look something like this:



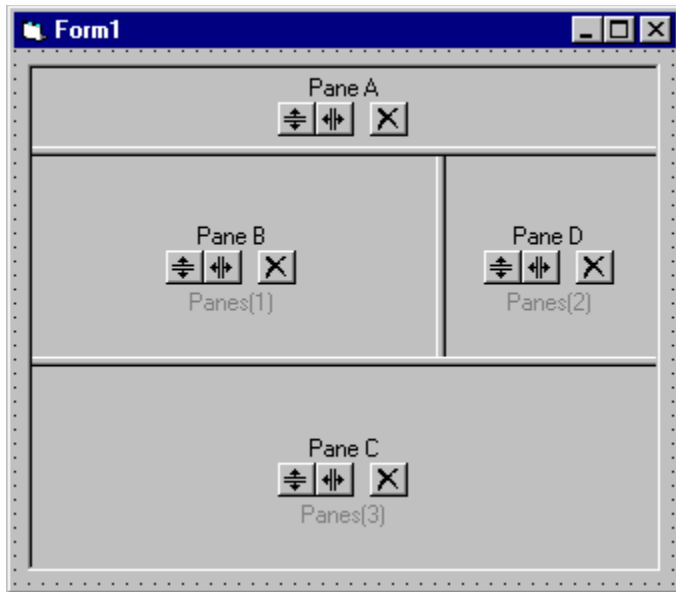
Pane A will eventually become the toolbar area of the application.

6. Now split Pane B horizontally. Pane B will be in the middle, with Pane C at the bottom of the control.



7. Click on the button of Pane B.
This will split the center of the control into two equal pieces, Pane B on the left and Pane D on the right.

8. Drag the splitter bar between B and D to the right until B occupies about two-thirds of the center area, and D occupies one-third. The control should now look something like this:



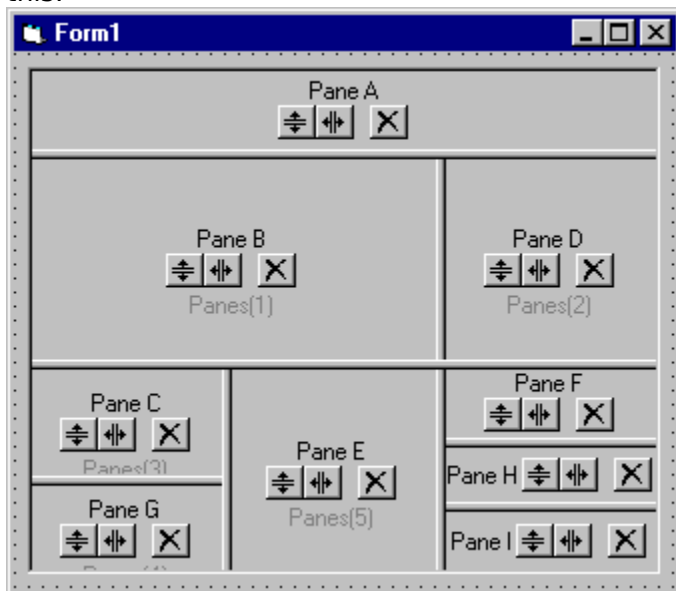
- Split Pane C vertically, creating C and E, then split Pane E vertically. The bottom area of the control should now be occupied by three panes - from left to right C, E and F. If you make any mistakes, simply click the

<input checked="" type="radio"/> Left Justified SSOption	Right Justified SSOption <input type="radio"/>
<input type="checkbox"/> Left Justified SSCheck	Right Justified SSCheck <input checked="" type="checkbox"/>

button on any pane you created

accidentally.

- Resize panes C, E and F so they are roughly the same width.
- Split Pane C horizontally, creating Pane G.
- Split Pane F horizontally to create Pane H, then split H horizontally to create I. Resize F, H and I so they are roughly the same height. At this point, your splitter should look like this:



- You will be using this project in the exercises that follow, so be sure to save it now.

Good work! You've successfully set up the basic structure of your application in just a few minutes, using nothing but your mouse.

In the next exercise, you will begin to add the individual controls that make up your application.

<input checked="" type="radio"/> Left Justified SSOption	Right Justified SSOption <input type="radio"/>
<input type="checkbox"/> Left Justified SSCheck	Right Justified SSCheck <input checked="" type="checkbox"/>

Exercise 2: Adding Single Controls To Panes

Now that you have created the panes you need to hold the controls of your application, its time to add the controls. You can begin with single controls, each of which will occupy a single pane.

Note Files covering all the concepts in this exercise, including any code you need to type, can be found in the Visual Basic project EX2.VBP, located under QUICK TOUR 2 in the SAMPLES subdirectory of your installation directory.

1. If you do not have the project from Exercise 1 available, open it now. Display Form1 which contains your Splitter.
2. The first control you will add to your application will be a text box. This is the central control of the application, as it provides note taking capabilities. Because it is the most important control, it will be located on the largest pane.

Select the standard Text Box control from your Visual Basic toolbox, and draw a text box inside of Pane B. The size of the text box does not matter, as long as it is drawn completely inside the pane.
3. As soon as you place the text box, the Splitter control resizes it to occupy all of Pane B. This gives you immediate feedback as to how the controls will look in the completed application.
4. Select a standard List Box from the toolbox and draw the list box inside of Pane D. Again, you will see the control automatically resized to fit the pane.

Note Because the height of the list box is constrained by the integral height of the list, the list box may not entirely fill the pane. This is normal behavior.

5. Add the following controls to the following panes. For each control, make sure you draw the control inside the boundaries of the pane. If you make a mistake, simply highlight the incorrectly placed control and delete it:

Pane C add SSCheck control (ActiveThreed check box)

Pane F add SSOption control (ActiveThreed option button)

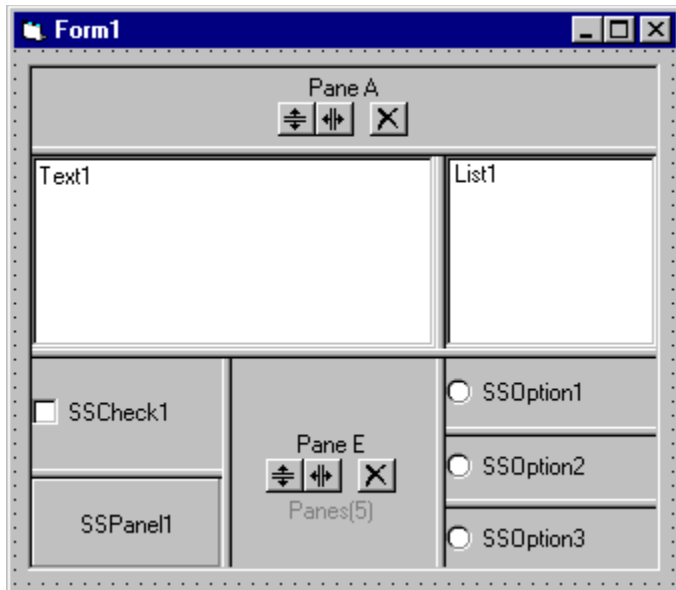
Pane G add SSPanel control (ActiveThreed panel)

Pane H add SSOption control (ActiveThreed option button)

Pane I add SSOption control (ActiveThreed option button)

(If the ActiveThreed THREED20 controls do not appear in your toolbox, open the Custom Controls dialog. When the list of available controls appears, check the box next to the "Sheridan ActiveThreed Controls" and click OK.)

When you are finished, the control should look like this:



Already the application is beginning to take shape. The List Box will provide a list of names or keywords that can be added to the notes area using only the mouse. The check box and option buttons will set various program options, and the SSPanel will display a digital clock.

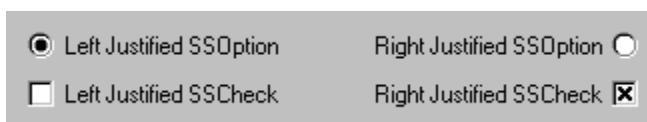
6. Grab the splitter bar between the list box and the text box, and drag it to the left. Notice how the controls smoothly and automatically resize to fit the new pane size. Also notice that the splitter bar between the text box and the list box operates independently of the splitter bar between Pane E and the three option buttons, even though they may look continuous. When you are done, drag the splitter bar back to it's previous position
7. Suppose you decide you want to give the digital clock the ability to add a time/date stamp to the notes area via drag and drop. For convenience, the SSPanel should be directly adjacent to the text box.

Moving controls around in the SSSplitter is as easy as adding them. To move the SSPanel to be underneath the text box, simply click on the panel and drag it. An outline of the control will appear. Drop the outline on top of the check box. The two controls automatically switch places.

8. Save the project at this stage. You may want to give the different stages of the project different file names.

Now you know how easy it is to add controls to the SSSplitter and move them around from pane to pane. You've seen the splitter in action as it resizes controls on the fly and swaps controls that you move with the mouse.

In the next exercise, you will see how to add multiple controls to a pane using container controls.



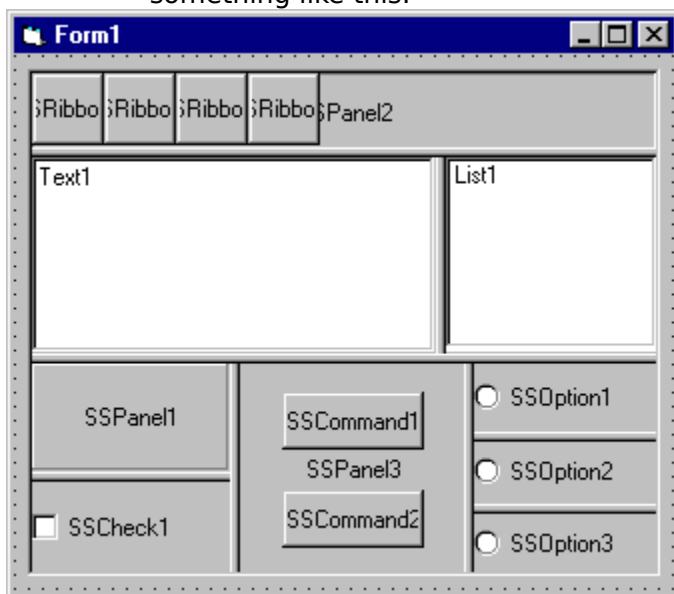
Exercise 3: Adding Multiple Controls To Panes

You have seen how to create Splitter panes, and how easy it is to add controls to panes. Each Splitter pane holds a single control, which is automatically resized to fill the area of the pane. However, there will probably be times when, to suit the layout of your application, you want more than one control in each pane.

To add multiple controls to a pane, you simply place a container control on the pane first. Any other controls can go "inside" the container. Although the individual controls will not be resized automatically, the container will be, and you can take appropriate action through code. You can use this as a technique if you want to place a single control inside a pane but not have it resize automatically; simply place a container control on the pane first, then place your fixed-size control inside the container.

Note Files covering all the concepts in this exercise, including any code you need to type, can be found in the Visual Basic project EX3.VBP, located under QUICK TOUR 2 in the SAMPLES subdirectory of your installation directory.

1. If you do not have the project from Exercise 2 available, open it now. Display Form1 which contains your Splitter.
2. Select an SSPanel control from the toolbox and draw a panel inside of Pane A. Set the **BevelOuter** property of the panel to '1 - Inset.'
3. Draw another SSPanel inside of Pane E. Set the **BevelOuter** property of this Panel to '0 - None.'
4. The pane at the top of the control will become the toolbar for the application. Select the SSRibbon button from the toolbox and draw a small Ribbon button on the topmost panel. The ribbon button should be about the same height as the panel and roughly square. Repeat this several times until you have four Ribbon buttons on the toolbar. Or you can copy the first button and paste it into the SSPanel three times. If you do this, do not create a control array.
5. Select the SSCommand button and add two command buttons to the SSPanel you just created in the bottom center pane. Your application should now look something like this:



6. The basic structure of the application is now complete. You would of course need to change the caption of each control and implement the application's functionality through code. You would probably also want to implement some restrictions on the sizing of Splitter panes, such as making sure the toolbar is always visible. You can do this using Splitter properties or through code, and these techniques are covered in the next exercise.
7. When you have multiple controls inside a container control, they are not automatically resized with the container. If you want to implement resizing of controls, you must do this through code. Select the toolbar panel (SSPanel2) and double click it to bring up the code window. Select the **Resize** event procedure.
8. There are several ways you might handle a resizable toolbar. The simplest is to simply resize and reposition the buttons based on the size of the SSPanel that contains them. Theoretically, this requires no changes to the Splitter, either at design time or through code, but realistically you would probably want to place a lower limit on the size of the pane.

In the **Resize** event procedure of the panel, enter the following code. (You do not need to type the comments, they are for your information only.)

```
' Test to see if the combined width of 4 buttons would
' exceed the width of the panel, and if so limit the
' first button's height to 1/4th of the panel's width
If SSPanel2.Height > (SSPanel2.Width / 4) Then
    SSRibbon1.Height = (SSPanel2.Width / 4)
Else
    SSRibbon1.Height = SSPanel2.Height
End If

' Make first button square and place in upper left corner
SSRibbon1.Width = SSRibbon1.Height
SSRibbon1.Left = 0
SSRibbon1.Top = 0

' Make second button same size as first and place it
' directly next to the first button
SSRibbon2.Height = SSRibbon1.Height
SSRibbon2.Width = SSRibbon1.Width
SSRibbon2.Left = SSRibbon1.Width
SSRibbon2.Top = 0

' Make third button same size as first and place it
' directly next to the second button
SSRibbon3.Height = SSRibbon1.Height
SSRibbon3.Width = SSRibbon1.Width
SSRibbon3.Left = (SSRibbon1.Width * 2)
SSRibbon3.Top = 0

' Make fourth button same size as first and place it
' directly next to the third button
SSRibbon4.Height = SSRibbon1.Height
SSRibbon4.Width = SSRibbon1.Width
SSRibbon4.Left = (SSRibbon1.Width * 3)
SSRibbon4.Top = 0
```

9. Save and run the project. Drag the topmost splitter bar up and down a few times

to see the effect on the SSRibbon buttons. As the Panel is resized, the buttons automatically adjust themselves to fit. However, they do not grow so large as to expand beyond the edges of the panel.

In the next exercise, you will see how to set limits on the size and movement of Splitter panes by using the properties of the Pane object. You will also be introduced to the Property Pages of the Splitter control, which give you access to properties unavailable through the Visual basic property sheet, such as the properties of sub-objects.

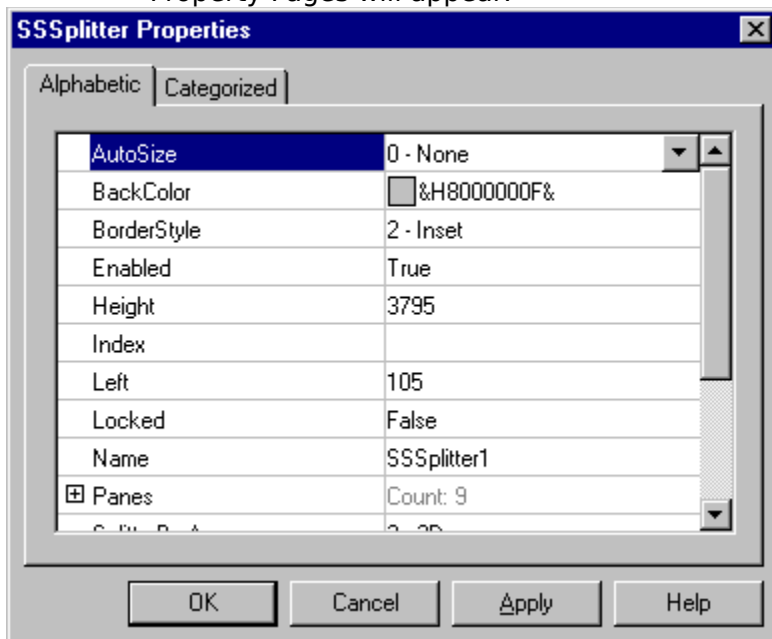
<input checked="" type="radio"/> Left Justified SSOption	Right Justified SSOption <input type="radio"/>
<input type="checkbox"/> Left Justified SSCheck	Right Justified SSCheck <input checked="" type="checkbox"/>


Exercise 4: Controlling Pane Size and Splitter Bar Movement

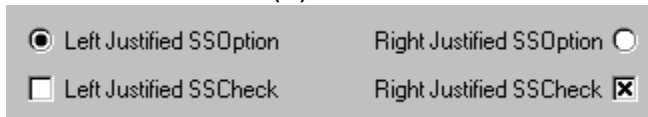
When setting up an application using the SSSplitter, you will often want to control the size of individual panes. Sometimes you may want a pane to have a minimum height or width, or you may want the size of the pane to remain constant. The Pane object has properties that make these tasks easy. To access the properties of the Pane object at design time, you must use the SSSplitter Property Pages.

Note Files covering all the concepts in this exercise, including any code you need to type, can be found in the Visual Basic project EX4.VBP, located under QUICK TOUR 2 in the SAMPLES subdirectory of your installation directory.

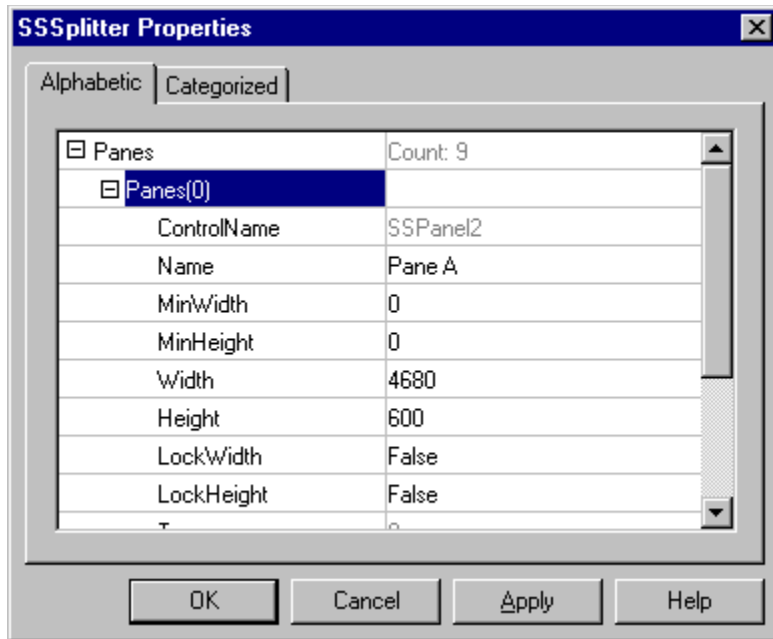
1. If you do not have the project from Exercise 3 available, open it now. Display Form1 which contains your Splitter.
2. Click on one of the splitter bars with the secondary mouse button to bring up the context menu for the SSSplitter control. Select 'Properties...' from the menu. The Property Pages will appear:



3. Select the item for the Panes collection, and click the  icon to expand the collection. You will see nine separate entries; one for each pane in the collection. You will be making two changes to the Splitter. You will set a minimum height for the pane that contains the toolbar (Pane A), and you will restrict the pane that contains the command buttons (Pane E) by locking its width and setting a minimum value for its height.
4. Select Panes(0) from the list and click on the

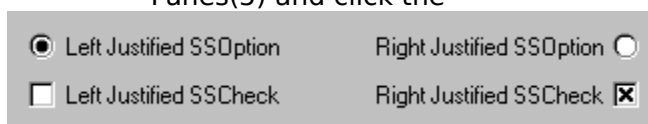


icon to expand the properties for the pane:



From here you can see the name of the control located on the pane, the pane's name and its other properties.

5. Select the **Name** property of the pane and change it from "Pane A" to the more descriptive "Toolbar." Hit Enter after you have entered the new name.
6. Two properties, **MinHeight** and **MinWidth**, control the minimum dimensions of the pane that the control will allow. You want to set the minimum height of the pane to a number large enough to ensure the toolbar will be visible and legible. Set the **MinHeight** property to 300 and hit Enter.
7. You may notice that as you make changes in the property pages, the properties whose values have changed are marked. Now click the Apply button. The values you have entered will be applied to the control, and the property pages will clear the marked properties, indicating that their values are now synchronized with those of the control. The property pages remain open for you to do more work.
8. Click the icon next to Panels(0) to collapse the properties for that pane. Select Panels(5) and click the



icon next to it to expand its properties. The

Name of the property should be Pane E and the **ControlName** property should be SSPanel3. Change the **Name** property of the pane to "Buttons".

9. Locking the dimensions of a pane is accomplished using the **LockHeight** and **LockWidth** properties. You want to lock the width of the pane at its current width, so set the **LockWidth** property to True. When you are done, click the Apply button.

Note The **LockHeight** and **LockWidth** properties only affect the pane at run time. At design time you can still change the height or width of the pane either by using the mouse or changing the **Height** and **Width** properties of the Pane object in the property pages.

10. Finally, you want to set the minimum height of this pane to a value just slightly

greater than the combined heights of the two command buttons on SSPanel3. The number you use depends on the size of your command buttons, but a value of 1000 should be about right. Enter 1000 for the **MinHeight** property of the pane, and click the OK button. The property pages will close.

11. Save and run the project. Try sizing the toolbar. You will see that when it reaches minimum size, you can no longer reduce its height. Similarly, the width of the panel containing the command buttons cannot be changed.
12. Stop the project. With the limits on pane size in place, you do not need to worry about losing the toolbar or clipping the command buttons. However, the command buttons may be incorrectly positioned when the SSPanel containing them is changed. To remedy this, enter the following code in the **Resize** event of the panel containing the command buttons (SSPanel3):

```
' Code for positioning command buttons vertically
' This code assumes both buttons are the same height
' Center Command1 in the top half of the panel and
' Command2 in the bottom half of the panel
Dim iHalfHeight As Integer
Dim iButtonTop As Integer

iHalfHeight = Int(SSPanel3.Height / 2)

iButtonTop = Int((iHalfHeight - SSCommand1.Height) / 2)

SSCommand1.Top = iButtonTop
SSCommand2.Top = iHalfHeight + iButtonTop
```

13. Save the project and run it again. This time when you resize the panel with the buttons, they stay evenly centered and spaced. Stop the project when you are finished.

Now you have seen how to limit and control panes through their properties, and written some more code for repositioning controls in response to the changing size of their container. In the next exercise, you will see how to write code that uses the events of the Splitter control to give you an even finer degree of control over the behavior of splitter panes.

<input checked="" type="radio"/> Left Justified SSOption	Right Justified SSOption <input type="radio"/>
<input type="checkbox"/> Left Justified SSCheck	Right Justified SSCheck <input checked="" type="checkbox"/>

Exercise 5: Accessing Panes Through Code


Sometimes you may want to have a greater degree of control over the splitter panes than simply locking them or setting their minimum dimensions. The SSSplitter provides you with several events and properties that let you control the panes through code.

Note Files covering all the concepts in this exercise, including any code you need to type, can be found in the Visual Basic project EX5.VBP, located under QUICK TOUR 2 in the SAMPLES subdirectory of your installation directory.

1. If you do not have the project from Exercise 4 available, open it now. Display Form1 which contains your Splitter.
2. The first feature you will implement is uniform resizing of the panes containing the option buttons. Currently, when the splitter bar above SSOption1 is moved, the pane containing SSOption1 is resized, but the panes below it are not. This affects the look of your application. To remedy this, you will write code to resize each pane when the top one is resized.

To make it easier to work with the panes which contain the option buttons, click on a splitter bar with your secondary mouse button and select "Properties..." from the context menu to invoke the property pages.



3. Click on the  icon next to the Panes collection. Change the names of the panes which hold the option buttons, Panes(6), Panes(7) and Panes(8), from "Pane F", "Pane H" and "Pane I" to "Opt1", "Opt2" and "Opt3," respectively.

While you have the property pages open, also change the name of Panes(3) from "Pane C" to "Clock." (This pane contains the SSPanel that will be the digital clock in the completed application.) This will save you a step later in the exercise. When you have changed the names of all four panes, click OK to accept the changes and close the property pages.

4. You will use the **Resize** event of the Splitter control to change the size of the option panes. **Resize** is triggered whenever a splitter bar is moved or when the control as a whole is resized. Double-click one of the splitter bars to bring up the code window for the SSSplitter, and enter the following code in the **Resize** event:

```
Dim lOptHeight As Long 'The height of one option panel

' Set the height of each option panel to be 1/3 the height
' of the Buttons panel, minus the approx. width of a splitter
bar
lOptHeight = Int(SSSplitter1.Panes("Buttons").Height / 3) - 50

' Set the pane unaffected by the splitter move to the correct
' height (avoids cascading events)
SSSplitter1.Panes("Opt3").Height = lOptHeight

' Set the topmost option pane to the correct height and the
' remaining pane will also be the right size
SSSplitter1.Panes("Opt1").Height = lOptHeight
```

4. Save and run the project. Try dragging the splitter bar below the text and list boxes up and down. You will see that all the panes containing the option buttons will stay the same size.
5. Currently, the pane containing the buttons is locked. This is good, because you cannot change its width and clip the edges of the buttons. But it is also bad because it constricts the movement of the panes on either side of it. Ideally, you would want to have the pane remain the same width, but still be moveable. You can do this through code, using the **Resize** event and the **SplitterStartDrag** event.

Double-click on one of the splitter bars to bring up the code window for the SSSplitter. Select the **SplitterStartDrag** event. The **SplitterStartDrag** event occurs when the splitter bar is first moved, before any resizing has taken place. It provides the following parameters:

- SplitterBarType** This is an integer value that tells you if the splitter bar being moved is a horizontal bar, a vertical bar, or a combination of the two.
- BorderPanes** This is a special collection of pane objects that includes only the panes affected by the movement of the splitter bar. You can step through the panes in this collection to see if a particular pane is among them, or to take action on each pane.
- Cancel** This is a boolean value that determines whether the movement of the splitter bar will succeed. It is normally False, but if you change it to True the splitter bar movement will be canceled.

First, enter the following code in the **(General) (declarations)** section of the form:

```
Private bJustOnce As Boolean
Private lOptWidth As Long
Private lButtonWidth As Long
Private lTrueButtonWidth As Long
```

6. Enter the following code for the **SplitterStartDrag** event. As always, comments are for your information and do not need to be typed:

```
' Set a flag to avoid cascading Resize events
bJustOnce = True

' Set widths to negative values so you can check them
' later to see which pane you're working with
lOptWidth = -1
lButtonWidth = -1

' Keep the real width of the Buttons pane
lTrueButtonWidth = SSSplitter1.Panes("Buttons").Width

' For each pane affected by the resize
For Each p In BorderPanes

    ' If it's an option pane (any one would do)
    If p.Name = "Opt3" Then
        ' Get its width and stop looking
        lOptWidth = SSSplitter1.Panes("Opt3").Width
```

```

Exit For
End If

' If it's the Buttons pane
If p.Name = "Buttons" Then
    ' Get it's width and keep looking
    ' to see if options pane is also affected
    lButtonWidth = SSSplitter1.Panes("Buttons").Width
End If
Next p

```

7. Next, enter the following code in the **Resize** event, after the code you previously entered there:

```

' Check flag to avoid cascading events
If bJustOnce = True Then
    bJustOnce = False

    ' If the width of the Buttons pane has changed
    ' but the width of the option pane has not
    ' (dragging splitter bar on left side of Buttons)
    If lOptWidth = -1 And lButtonWidth <> -1 Then
        ' Find out how much it has changed
        lBWidthChange = SSSplitter1.Panes("Buttons").Width -
lButtonWidth
        ' Add the amount of change to the width of the option pane
        ' To push the right edge of the Buttons pane into place
        SSSplitter1.Panes("Opt3").Width =
SSSplitter1.Panes("Opt3").Width + lBWidthChange

        ' But if the widths of both the Button and option
        ' panes have changed (dragging splitter on right
        ' side of Buttons)
        ElseIf lButtonWidth <> -1 And lOptWidth <> -1 Then
            ' Find out how much the width of the option pane has changed
            lOWidthChange = lOptWidth - SSSplitter1.Panes("Opt3").Width
            ' And add that amount to the width of the Clock pane
            ' To push the left edge of the Buttons pane into place
            SSSplitter1.Panes("Clock").Width =
SSSplitter1.Panes("Clock").Width + lOWidthChange
        End If

        ' Finally, check the width of the Buttons panel and
        ' set it to the correct value
        If SSSplitter1.Panes("Buttons").Width <> lTrueButtonWidth Then
            SSSplitter1.Panes("Buttons").Width = lTrueButtonWidth
        End If
    End If

End If

```

8. The last thing you must do is bring up the property pages for the Splitter, select Panes(5) and set the **LockWidth** property of the "Buttons" pane to False. This will make the Buttons pane moveable, while the code you entered will make sure it remains the correct width.

9. Save and run the project. You will see that when you drag the splitter on the left or the right of the buttons panel, the pane moves, but retains its size. The panes on the opposite side of the panel are resized accordingly.

Congratulations! You have now completed all of Quick Tour 2. You have seen how to use the SSSplitter to organize controls in resizable panes, both singly and in groups. You have explored the property pages and used them to set the properties of objects. And you have gained an understanding of how to control pane size and movement through code.

If you have any further questions on using the SSSplitter, you can consult the Control Reference to get more specific help with individual properties and events. If you want to learn more about the property pages, see the section entitled **Property Pages**.

<input checked="" type="radio"/> Left Justified SSOption	Right Justified SSOption <input type="radio"/>
<input type="checkbox"/> Left Justified SSCheck	Right Justified SSCheck <input checked="" type="checkbox"/>

Files Property

[See Also](#)

[Applies To](#)

Description

Returns an **ssDataObjectFiles** collection, which in turn contains a list of all filenames used by an **ssDataObject** object (such as the names of files that a user drags to or from the Windows Explorer.)

Syntax

object.**Files**(*index*)

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>index</i>	An integer expression which is an index to an array of filenames.

Remarks

The **ssDataObjectFiles** collection is filled with filenames only when the **ssDataObject** object contains data of type ssCFFiles (see **Constants** for more information. The **ssDataObject** object can contain several different types of data.) You can iterate through the collection to retrieve the list of file names.

The **ssDataObjectFiles** collection can be filled to allow ActiveThreed controls to act as a drag source for a list of files.

FloodColor Property

[See Also](#)

[Example](#)

[Applies To](#)

Description

Returns or sets the color used to paint the area inside the panel's inner bevel when the panel is used as a status or progress indicator (when the **FloodType** property setting is other than None.)

Syntax

object.**FloodColor** [= *color*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>color</i>	A value or constant that determines the color of the specified object.

Remarks

Use this property with **FloodFillStyle**, **FloodPercent**, **FloodShowPct** and **FloodType** to cause the panel to display a colored status bar indicating the degree of completion of a task.

FloodFillStyle Property

[See Also](#)

[Example](#)

[Applies To](#)

Description

Returns or sets a value that specifies the appearance of the panel when used as a status or progress indicator.

Syntax

object.**FloodFillStyle** [= *number*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the style of the flood fill, as described in Settings.

Settings

Setting	Description
0	(Default) Solid. The flood fill will display as a continuous colored area.
1	Segmented. The flood fill will display as evenly divided colored segments.

Remarks

Use this property with **FloodColor**, **FloodPercent**, **FloodShowPct** and **FloodType** to cause the panel to display a colored status bar indicating the degree of completion of a task.

FloodPercent Property

[See Also](#)

[Example](#)

[Applies To](#)

Description

Returns or sets the percentage of the painted area inside the panel's inner bevel when the panel is used as a status or progress indicator (when the **FloodType** property setting is other than None).

Syntax

object.**FloodPercent** [= *number*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression between 0 and 100 specifying the amount (percentage) of flood fill.

Remarks

The **FloodPercent** property can be set to an integer value between 0 and 100.

Use this property in conjunction with **FloodColor**, **FloodFillStyle**, **FloodShowPct**, and **FloodType** to cause the panel to display a colored status bar, indicating the degree of completion of a task.

This property is not available at design time.

FloodShowPct Property

[See Also](#)

[Example](#)

[Applies To](#)

Description

Returns or sets a value that determines whether the current setting of the **FloodPercent** property will be displayed in the center of the panel when the panel is used as a status or progress indicator (when the **FloodType** property setting is other than None).

Syntax

object.**FloodShowPct** [= *boolean*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying whether the panel will display a flood fill, as described in Settings.

Settings

Setting	Description
True	The current setting of the FloodPercent property will be displayed.
False	(Default) The current setting of the FloodPercent property will not be displayed.

Remarks

Use this property with **FloodColor**, **FloodPercent**, **FloodShowPct** and **FloodType** to cause the panel to display a colored status bar indicating the degree of completion of a task.

FloodType Property

[See Also](#)

[Example](#)

[Applies To](#)

Description

Returns or sets a value that specifies the appearance of the panel when used as a status or progress indicator.

Syntax

object.**FloodType** [= *number*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the type of flood fill, as described in Settings.

Settings

Setting	Description
0	(Default) None. Panel has no status bar capability and the caption (if any) is displayed.
1	Left to right. Panel will be painted in a color, which is specified by the FloodColor property, from the left inner bevel to the right as the FloodPercent property increases.
2	Right to left. Panel will be painted in a color, which is specified by the FloodColor property, from the right inner bevel to the left as the FloodPercent property increases.
3	Top to bottom. Panel will be painted in a color, which is specified by the FloodColor property, from the top inner bevel downward as the FloodPercent property increases.
4	Bottom to top. Panel will be painted in a color, which is specified by the FloodColor property, from the bottom inner bevel upward as the FloodPercent property increases.
5	Widening circle. Panel will be painted in a color, which is specified by the FloodColor property, from the center outward in a widening circle as the FloodPercent property increases.

Remarks

Use this property with **FloodColor**, **FloodFillStyle**, **FloodPercent** and **FloodShowPct** to cause the panel to display a colored status bar indicating the degree of completion of a task.

Font Object

[See Also](#)

[Applies To](#)

Description

Contains the information used to format the display of text in the caption of the control.

Properties

[Bold](#)

[Italic](#)

[Name](#) (default)

[Size](#)

[Strikethrough](#)

[Underline](#)

Remarks

You frequently identify a Font object using the **Font** property of an object that displays text.

Font Property

[See Also](#)

[Applies To](#)

Description

Returns or sets the properties of the Font object at design time and run time.

Syntax

object.**Font**

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.

Remarks

This setting provides access to the properties of the **Font** object. At design time, you can choose this setting to invoke the Font dialog box and set the font attributes of the object. At run time, specify a property setting of the object that you want to change or return.

Font3D Property

[See Also](#)

[Applies To](#)

Description

Returns or sets a value that specifies the 3-D style of the control's caption text.

Syntax

object.**Font3D** [= *number*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the 3-D font style, as described in Settings.

Settings

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 above the background.
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 background.
4	Inset w/heavy shading. Caption appears even more inset.
5	Drop Shadow. Caption appears with a dark gray drop shadow slightly below and to the right of the text.

Remarks

The **Font3D** property works in conjunction with all the other font properties (bold, italic, etc.) 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.

ForeColor Property

[See Also](#)

[Applies To](#)

Description

Returns or sets the foreground (text) color of the control.

Syntax

object.**ForeColor** [= *color*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>color</i>	A value or constant that determines the color of the specified object.

GetData Method

[See Also](#)

[Applies To](#)

Description

This method is used to return data from an **ssDataObject** object in the form of a variant.

Syntax

object.**GetData** (*format*)

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>format</i>	An integer expression specifying the data format, as described in Settings. Parentheses must enclose the constant or value. If format is 0 or omitted, GetData automatically uses the appropriate format.

Settings

Setting	Description
1	Text (.TXT files)
2	Bitmap (.BMP files)
3	metafile (.WMF files)
8	Device-independent bitmap (DIB)
9	Color palette
14	Enhanced metafile (.EMF files)
15	List of files
-16639	Rich text format (.RTF files)

Remarks

Constants that correspond to these formats are available. See [Constants](#) for more information.

It's possible for the **GetData** and **SetData** methods to use data formats other than those listed in Settings, including user-defined formats registered with Windows via the RegisterClipboardFormat() API function. However, there are a few caveats:

- § The **SetData** method requires the data to be in the form of a byte array when it does not recognize the data format specified.
- § The **GetData** method always returns data in a byte array when it is in a format that it doesn't recognize, although Visual Basic can transparently convert this returned byte array into other data types, such as strings.
- § The byte array returned by **GetData** will be larger than the actual data when running on some operating systems, with arbitrary bytes at the end of the array. The reason for this is that the application does not always know the data's format, and knows only the amount of memory that the operating system has allocated for the data. This allocation of memory is often larger than is actually required for the data. Therefore, there may be extraneous bytes near the end of the allocated memory segment. As a result, you must use appropriate functions to interpret the returned data in a meaningful way (such as truncating a string at a particular length with the Left function if the data is in a text format).

Note Not all applications support ssCFBitmap or ssCFPalette, so it is recommended that

you use ssCFDIB whenever possible.

GetFormat Method

[See Also](#)

[Applies To](#)

Description

This method returns a Boolean value indicating whether an item in the **ssDataObject** object matches a specified format. This method does not support named arguments.

Syntax

object.**GetFormat** *format*

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>format</i>	An integer expression specifying the data format, as described in Settings.

Settings

Setting	Description
1	Text (.TXT files)
2	Bitmap (.BMP files)
3	metafile (.WMF files)
8	Device-independent bitmap (DIB)
9	Color palette
14	Enhanced metafile (.EMF files)
15	List of files
-16639	Rich text format (.RTF files)

Remarks

Constants that correspond to these formats are available. See [Constants](#) for more information.

The **GetFormat** method returns True if an item in the **ssDataObject** object matches the specified format. Otherwise, it returns False.

GroupAllowAllUp Property

[See Also](#)

[Applies To](#)

Description

Returns or sets a value that determines whether or not all buttons in a group can be in the 'up' position.

Syntax

object.**GroupAllowAllUp**[= *boolean*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying state of the buttons in the group, as described in Settings.

Settings

Setting	Description
True	All buttons in the current group may be in the 'up' position.
False	(Default) At least one button in the current group must be depressed.

Note When this property is set for one button in a group, then the property is automatically set to the same value for all the other buttons in the group.

Remarks

This property controls the behavior of ribbon buttons in an exclusive group, as specified by their **GroupNumber** property. Normally, when buttons are in the same group, one of the buttons is pushed in at all times. By setting this property to True, you can override this behavior, so it becomes possible that none of the buttons in the group will be in the 'down' state.

GroupNumber Property

[See Also](#)

[Applies To](#)

Description

Returns or sets the number of the group to which the button belongs.

Syntax

object.**GroupNumber** [= *number*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the number of the button's group.

Remarks

You can assign ribbon buttons to a group by setting the **GroupNumber** property of the buttons to the same value. Buttons in the same group exhibit exclusive behavior, effectively operating like option buttons. Depressing one button in a group automatically causes the others to return to the undepressed state

By setting the **GroupAllowAllUp** property, you can specify whether the button group requires one button to be depressed at all times.

Height Property

[See Also](#)

[Applies To](#)

Description

Returns or sets the vertical dimension of an object.

Syntax

object.**Height** [= *number*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An single precision expression specifying the height of the object. The value returned is in the scale units of the object's container.

Remarks

This is an [extender](#) property, except when applied to the **Pane** object.

Use the **Height**, **Width**, **Left**, and **Top** properties for operations or calculations based on an object's total area, such as sizing or moving the object.

HelpContextID Property

Applies To

Description

Returns or sets an associated context number for an object. Used to provide context-sensitive help for your application.

Syntax

object.**HelpContextID** [= *number*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the context number of the Help topic associated with the object, as described in Settings..

Settings

Setting	Description
0	(Default) No context number specified.
> 0	An integer specifying a valid context number.

Remarks

This is an [extender](#) property.

For context-sensitive help on an object in your application, you must assign the same context number to both object and to the associated help topic when you compile your help file.

hWnd Property

Applies To

Description

Returns the window handle of the control.

Syntax

object.**hWnd**

Remarks

Note The ActiveThread non-container controls (all except SSFrame & SSPanel) normally operate as windowless controls in any environment that supports windowless controls. When a control is windowless, using the **hWnd** property to return the window handle **will create a window** for the control.

This property is read-only and is only available at run time.

Included Files

The following table gives a brief description of the files that are installed on your hard disk during the Setup process.

Filename(s)	Description
THREED20.OCX	ActiveThread file that contains the SSCheck, SSCommand, SSFrame, SSOOption, SSPanel, and SSRibbon ActiveX controls.
SPLITTER.OCX	ActiveThread file that contains the SSSplitter ActiveX control.
THREED20.HLP THREED20.CNT	ActiveThread on-line help files.
README.WRI	Pertinent, up-to-date version information on ActiveThread, plus additions and corrections to the documentation
HISTORY.WRI	Revision history of ActiveThread outlining changes to the product from version to version.
UPGRADE.TXT	Text file that contains information on how to upgrade applications that currently use THREED.VBX, THREED16.OCX and THREED32.OCX.
\SAMPLES*.*	Samples and exercise files provided with ActiveThread. Also contains the project files used by the Quick Tour tutorials
\GRAPHICS*.*	Bitmaps and animations provided for use with ActiveThread in your applications.
SSPPG.DLL	Property Page DLL for design-time support.
MSVCRT.DLL OLEPRO32.DLL OLEAUT32.DLL STDOLE2.TLB	Support files required by ActiveX controls
UNINSTAL.EXE	Uninstall program used to uninstall ActiveThread. It is recommended that you use the facility provided by your operating system to uninstall applications, if one is available. Windows 95 and Windows NT 4.0 users should use the "Add/Remove Programs" tool in Control Panel to uninstall ActiveThread.
INSTALL.LOG	Log file created by the install program and used by UNINSTAL.EXE

Including Sound

Sound is a big part of producing multimedia applications. Sound support for the ActiveThread controls is implemented in a simple and straightforward manner. Each control supports a **PlaySoundFile** method. When calling this method, you pass it the name of the sound file you would like to be played. Sound files must be in the Windows WAV format. The method can be called from any event, function or subroutine in your application.

Note The computer running the application must have a properly installed and configured audio playback device in order to use sounds.

Unless you specify a fully qualified pathname for the **PlaySoundFile** method, the ActiveThread controls will look for the sound file first in the same directory as your executable, then in any directories specified by the environment's PATH statement. This makes it easy to distribute sound files with your completed application - simply place them in the same directory as the executable. However, at design time, the executable used is that of your design environment. Any sounds you wish to use must be placed in the same directory as the executable for your programming language. This is only necessary until you are ready to compile and distribute the application.

To add sound to an event in an ActiveThread control:

1. Bring up the code window for the control, and select the name of the event that you want to trigger the sound.
2. In the event procedure, enter the following code, substituting the name of the control for SSControl and the name of your chosen sound file for MYWAVE.WAV:

```
SSControl.PlaySoundFile "MYWAVE.WAV"
```

3. Make sure the sound file is located in a directory where it can be found by the control; either the same directory as your development environment's executable, or one of the directories along your path.

Quick Tour 1 Introduction



ActiveThread is a set of ActiveX controls designed to add a cutting-edge interface to your applications. The ActiveThread THREED20 controls (THREED20 refers to all the ActiveThread controls **except** the SSSplitter) replicate the functions of some of the Windows base controls, but with a variety of cool new features. The THREED20 controls give you the tools to design attractive and engaging applications, utilizing many of the interface features popularized by the Internet and the World Wide Web. Together with the [Splitter](#) control, the THREED20 controls provide enhanced interface capabilities which you can use to give your application a multimedia look and feel. You can design programs that integrate well with content on the World Wide Web as well as the most recent versions of popular applications suites.

Note The code and forms created in these exercises are included with the product. They are in the SAMPLES\QUICK TOUR 1 subdirectory of your product installation directory.

In the code included with these exercises, the ActiveThread constants are used wherever applicable. It is recommended that you use the constants in your own code, rather than the numeric values, to insure compatibility with future versions of ActiveThread. Constants for the various enumerated properties can be found in the Object Browser.

Lesson 1: General Properties

This Quick Tour will introduce you to the primary features of the THREED20 controls; SSCheck, SSCommand, SSFrame, SSOption, SSPanel and SSRibbon.

ActiveThread's THREED20 controls have been designed to be easy to use. They share many of the same properties and methods from control to control. Once you learn how to work with the interface features of one control, you can apply your knowledge to any of the THREED20 controls. In this Quick Tour, you will first learn how to use the general features of the THREED20 controls, then move on to more control-specific features.

ActiveThread THREED20 controls can be used in a variety of development environments. For the sake of simplicity, the Quick Tours will use the Visual Basic 4.0 development platform to illustrate the features of the controls. Examples of how to use the controls in other environments are included in the SAMPLES directory, which is located in the subdirectory where you installed ActiveThread.

<input checked="" type="radio"/> Left Justified SSOption	Right Justified SSOption <input type="radio"/>
<input type="checkbox"/> Left Justified SSCheck	Right Justified SSCheck <input checked="" type="checkbox"/>

Quick Tour 2 Introduction

Left Justified SSOption Right Justified SSOption
 Left Justified SSCheck Right Justified SSCheck

ActiveThread includes a totally new control called the SSSplitter control. The Splitter is a flexible container control that allows you to organize multiple controls in a series of resizable panes. When you drag the splitter bars that separate these panes with your mouse, the control contained by each pane is automatically resized. Each pane can contain only one control, but that control may be a container control such as the SSPanel, which gives you the option of putting multiple controls in a single pane. The SSSplitter control brings the frames metaphor, found on many web sites, to your desktop applications.

Aside from the splitter bars, the Splitter has no run-time user interface, since the control is completely covered with other controls. The design time interface of the splitter is simple and intuitive - on screen buttons make it easy to delete panes or add new panes by splitting existing ones horizontally or vertically. Similarly, the Splitter has just a few events to deal with the movement of splitter bars and the resizing of panes. Despite its simplicity, the Splitter is quite flexible and powerful.

Controls are added to the Splitter the same way you would add them to any other container control. Once inside, controls can be moved from pane to pane via simple drag and drop.

Note The code created in these exercises is included with the product. It is in the `SAMPLES\QUICK TOUR 2` subdirectory of your Product installation directory.

Learning to use the Splitter Control

In this Quick Tour, you will construct a simple framework application using the Splitter. The application will be a simple Personal Information Manager (PIM) type of program featuring many of the elements you would use in creating real world applications, such as data entry fields, toolbars, list boxes and various types of buttons. Along the way, you will fully explore the capabilities of the Splitter control. For the sake of simplicity, the Quick Tours will use the Visual Basic 4.0 development platform to illustrate the features of the control.

More so than in Quick Tour 1, you may want to rely on the samples provided with ActiveThread as you move through the exercises, because of the cumulative nature of the work you will be doing.

Left Justified SSOption Right Justified SSOption
 Left Justified SSCheck Right Justified SSCheck

Italic Property

[See Also](#)

[Applies To](#)

Description

Returns or sets the font style of the specified **Font** object to either italicized or non-italicized.

Syntax

object.**Italic** [= *boolean*]

The **Italic** property syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A Boolean expression specifying the font style, as described in Settings.

Settings

The settings for *boolean* are:

Setting	Description
True	Turns on italic formatting.
False	Turns off italic formatting.

Remarks

The **Font** object is not directly available at design time. Instead you set the **Italic** property through a control's **Font** property.

At run time, however, you can set **Italic** directly by specifying its setting for the **Font** object.

Item Method

[Applies To](#) [Example](#)

Description

Returns the pane object corresponding to the specified name or index.

Syntax

object.**Item** *name*

object.**Item**(*index*)

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list..
<i>name</i>	A string expression that evaluates to the name of an existing Pane object.
<i>index</i>	An integer expression that evaluates to the position of a Pane object within the Panes collection.

Remarks

This is the default method of the **Panes** collection. Individual panes within the collection may be accessed using this method

If *index* is a numeric expression, it must be a number from 1 to the value of the collection's Count property. If the value provided as index does not match any existing member of the collection, an error occurs.

KeyDown Event

[See Also](#)

[Applies To](#)

Description

Occurs when the user presses a key while an object has the focus.

Syntax

Sub *control_KeyDown* ([*index As Integer*] *keycode As Integer*, *shift As Integer*)

The event parameters are:

Parameter Description

<i>index</i>	An integer expression that uniquely identifies a control if it is in a control array.
<i>keycode</i>	A key code, such as 112 (the F1 key) or 36 (the HOME key.) Your development environment may have pre-defined constants for key code values; you should use them wherever possible.
<i>shift</i>	An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys at the time of the event.

Remarks

The *shift* argument is a bit field with the least-significant bits corresponding to the SHIFT key (bit 0), the CTRL key (bit 1), and the ALT key (bit 2). These bits correspond to the values 1, 2, and 4, respectively. The *shift* argument indicates the state of these keys. Some, all, or none of the bits can be set, indicating that some, all, or none of the keys are pressed. For example, if both CTRL and ALT are pressed, the value of *shift* is 6.

KeyDown is *not* invoked for:

- § The ENTER key if the control is an SsCommand control with the **Default** property set to True.
- § The ESC key if the control is an SsCommand control with the **Cancel** property set to True.

KeyPress Event

[See Also](#)

[Applies To](#)

Description

Occurs when the user presses and releases a key

Syntax

Sub *control_KeyPress* ([*Index As Integer*] *keyascii As Integer*)

The event parameters are:

Parameter Description

<i>Index</i>	An integer expression that uniquely identifies a control if it is in a control array.
<i>keyascii</i>	An integer that returns a standard numeric ANSI keycode. <i>Keyascii</i> is passed by reference; changing it sends a different character to the object. Changing <i>keyascii</i> to 0 cancels the keystroke so the object receives no character.

Remarks

In order to generate a KeyPress event, the key pressed must be an ANSI key: any printable keyboard character, the CTRL key combined with a character from the standard alphabet or one of a few special characters, and the ENTER or BACKSPACE key. Changing the value of the *keyascii* argument changes the character displayed.

KeyUp Event

[See Also](#)

[Applies To](#)

Description

Occurs when the user releases a key after having pressed it while an object has the focus.

Syntax

Sub *control_KeyUp* ([*index As Integer*] *keycode As Integer*, *shift As Integer*)

The event parameters are:

Parameter Description

<i>index</i>	An integer expression that uniquely identifies a control if it is in a control array.
<i>keycode</i>	A key code, such as 112 (the F1 key) or 36 (the HOME key.) Your development environment may have pre-defined constants for key code values; you should use them wherever possible.
<i>shift</i>	An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys at the time of the event. A bit is set if the key is down.

Remarks

The *shift* argument is a bit field with the least-significant bits corresponding to the SHIFT key (bit 0), the CTRL key (bit 1), and the ALT key (bit 2). These bits correspond to the values 1, 2, and 4, respectively. The *shift* argument indicates the state of these keys. Some, all, or none of the bits can be set, indicating that some, all, or none of the keys are pressed. For example, if both CTRL and ALT are pressed, the value of *shift* is 6.

KeyDown is *not* invoked for:

- § The ENTER key if the control is an SsCommand control with the **Default** property set to True.
- § The ESC key if the control is an SsCommand control with the **Cancel** property set to True.

Learning To Use ActiveThread Controls

Introduction to Quick Tour 1

Lesson 1: General Properties

- Exercise 1: Adding a scrolling caption
- Exercise 2: Playing sounds
- Exercise 3: Transparent operation
- Exercise 4: Adding an animated picture
- Exercise 5: Mask colors & AutoSize

Lesson 2: SSCommand & SSRibbon

- Exercise 6: Border styles & button pictures
- Exercise 7: Adding additional button states
- Exercise 8: Using buttons in groups

Lesson 3: SSCheck & SSOption

- Exercise 9: Captions, pictures and alignment
- Exercise 10: Adding a custom control graphic
- Exercise 11: Working with control states (2 vs. 3)

Lesson 4: SSFrame & SSPanel

- Exercise 12: Understanding background pictures
- Exercise 13: More on alignment
- Exercise 14: Making a progress indicator

Back To Quick Tours

Learning To Use The Splitter Control

Introduction to Quick Tour 2

Exercise 1: [Setting up Splitter panes](#)

Exercise 2: [Adding single controls to panes](#)

Exercise 3: [Adding multiple controls to panes](#)

Exercise 4: [Controlling pane size and splitter bar movement](#)

Exercise 5: [Accessing panes through code](#)

Back To Quick Tours

Left Property

[See Also](#)

[Applies To](#)

Description

Returns or sets the distance between the left edge of an object and the left edge of its container.

Syntax

object.**Left** [= *number*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An single precision expression specifying the distance of the object from the left edge of its container. The value returned is in the scale units of the object's container.

Remarks

This is an [extender](#) property, except when applied to the **Pane** object.

Use the **Height**, **Width**, **Left**, and **Top** properties for operations or calculations based on an object's total area, such as sizing or moving the object.

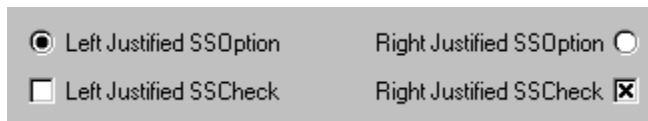
This property is read-only the **Pane** object.

Lesson 2: SSCommand & SSRibbon

Now that you are familiar with the properties and methods common to all the THREEED20 controls, it is time to begin exploring each individual control in further detail. Each control has unique abilities that set it apart from the rest.

This lesson covers properties exclusive to the SSCommand and SSRibbon buttons. Although both are buttons, the SSCommand operates like the Windows common command button - once clicked, it automatically pops back up. The SSRibbon is a toggle button that behaves in a manner similar to a check box or option button. Once clicked, the button changes state, staying pressed down. A second click is required to pop it back up.

SSRibbon buttons also have the ability to operate in exclusive mode, like option buttons. The control automatically takes care of ensuring that no two buttons in the same group can be pressed at the same time.

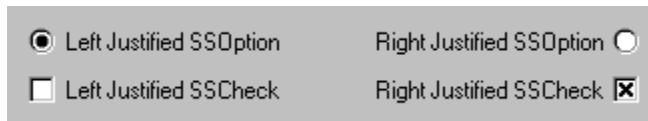


Lesson 3: SSCheck & SSOption

Lessons 1 of Quick Tour 1 introduced the THREED20 controls (the controls contained in THREED20.OCX) and taught how to use the features that are common to all the controls. Lesson 2 covered two specific controls, the SSCommand and SSRibbon button controls.

Lesson 3 builds on the knowledge gained in Lessons 1 and 2, and goes into deeper detail on two other THREED20 controls; SSCheck and SSOption. These two controls have much in common with one another, and both feature the common ActiveThreed ability to create a truly customized interface experience for the users of your programs.

This lesson will introduce you to some of the nuances of using scrolling captions with the Check and Option controls. You will also find out more about control states, and use that knowledge to create custom graphics that replace the "Square Box - Round Box" look of the standard Check and Option controls. Finally, you will discover how to use the different states of the SSCheck control to ensure compatibility with both older and newer programs.



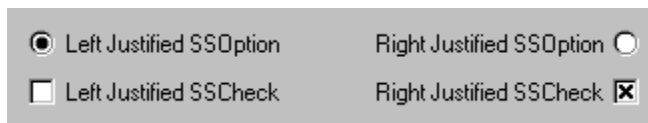
Lesson 4: SSFrame & SSPanel

This final lesson of Quick Tour 1 covers the two remaining THREED20 controls, SSFrame and SSPanel. As with the previous lessons, it will build on the knowledge you have already acquired.

The primary purpose of the SSFrame and SSPanel controls is to act as containers for other controls. In addition to grouping controls functionally, SSFrame and SSPanel allow you to group controls visually with extended interface properties.

Adding controls to an SSPanel or SSFrame control is identical to adding controls to the Windows common Picture Box or Frame container controls. Simply place the SSFrame or SSOption on a form, then draw the controls inside the container control. Alternatively, you can cut controls from a form to the clipboard, select the SSFrame or SSPanel, then paste the controls inside.

The SSPanel control can also serve as a flexible label-type control, displaying information either textually or graphically.



LockHeight Property

[See Also](#)

[Applies To](#)

Description

Returns or sets a value that determines whether the height of the pane can be changed

Syntax

object.**LockHeight** [= *boolean*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying whether the pane may be vertically resized, as described in Settings.

Settings

Setting	Description
True	The height of the pane cannot be changed.
False	(Default) The pane height may be changed by moving the splitter bars.

Remarks

You can use this property to ensure that a particular pane always remains the same height.

Note that this limitation applies only to resizing the pane using the splitter bars. You may still set the pane to any size through code, regardless of the value of this property.

This property takes effect at run time. At design time you can move the splitter bars without restriction.

LockWidth Property

[See Also](#)

[Applies To](#)

Description

Returns or sets a value that determines whether the width of the pane can be changed

Syntax

object.**LockWidth** [= *boolean*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying whether the pane may be horizontally resized, as described in Settings.

Settings

Setting	Description
True	The width of the pane cannot be changed.
False	(Default) The pane width may be changed by moving the splitter bars.

Remarks

You can use this property to ensure that a particular pane always remains the same width.

Note that this limitation applies only to resizing the pane using the splitter bars. You may still set the pane to any size through code, regardless of the value of this property.

This property takes effect at run time. At design time you can move the splitter bars without restriction.

Locked Property

[See Also](#)

[Applies To](#)

Description

Returns or sets a value that determines whether the splitter bars will be movable or stationary.

Syntax

object.**Locked** [= *boolean*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying whether the splitters between panes may be moved, as described in Settings.

Settings

Setting	Description
True	Splitters cannot be moved and panes cannot be resized.
False	(Default) Splitters can move and panes are sizable.

Remarks

If you are designing an application with a fixed layout, you can use this property to prevent the user from moving the splitter bars in the Splitter control. Normally, the splitter bars are movable, allowing the user to resize panes.

This property takes effect at run time. At design time you can move the splitter bars without restriction.

MarqueeCycleBegin Event

[See Also](#)

[Example](#)

[Applies To](#)

Description

Occurs just before the control's caption begins a new marquee cycle.

Syntax

Sub *control*_**MarqueeCycleBegin** ([*index* **As Integer**])

The event parameters are:

Parameter Description

<i>index</i>	An integer expression that uniquely identifies a control if it is in a control array.
--------------	---

Remarks

The **MarqueeCycleBegin** event occurs before the caption text appears in a control, when the text has finished scrolling out of the caption area and is about to re-appear (Scrolling Marquee), when the text has reached an edge of the control and is about to reverse direction (Bouncing Marquee), when the text has reached the point where it has come to a stop (Sliding Marquee), or when the text is about to become visible again (Blinking Marquee.)

MarqueeCycleEnd Event

[See Also](#)

[Applies To](#)

Description

Occurs after the control's caption completes a marquee cycle.

Syntax

Sub *control*_**MarqueeCycleEnd** ([*index* **As Integer**])

The event parameters are:

Parameter Description

<i>index</i>	An integer expression that uniquely identifies a control if it is in a control array.
--------------	---

Remarks

The **MarqueeCycleEnd** event is similar to the **MarqueeCycleBegin** event and usually occurs at nearly the same time, although slightly before. It's primary purpose is for use with the Sliding marquee style, as it gives you a way to take an action when the motion of the text has stopped.

MarqueeDelay Property

[See Also](#)

[Example](#)

[Applies To](#)

Description

Returns or sets the scrolling speed of the marquee effect of the caption text.

Syntax

object.**MarqueeDelay** [= *number*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the delay between scrolling effects in milliseconds..

Remarks

This property controls the speed of the text scrolling. Setting **MarqueeDelay** to 0 will cause the control to scroll the text as quickly as possible. Setting any value for this property will slow down scrolling by introducing a delay into the amount of time it takes the control to scroll the caption one unit in any direction. The size of a unit is determined by the **MarqueeScrollAmount** property.

Use this property in conjunction with the **MarqueeScrollAmount** property to achieve the desired scrolling speed and smoothness.

Warning! Setting MarqueeDelay to 0 or a low number (less than 55 for Windows 95, less than 10 for Windows NT) will increase the demands the control places on the system's CPU. Depending on the machine running the application, this can have a significant effect on overall system performance.

MarqueeDirection Property

[See Also](#)

[Example](#)

[Applies To](#)

Description

Returns or sets the scrolling direction of the marquee effect of the caption text.

Syntax

object.**MarqueeDirection** [= *number*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the type of marquee scrolling to perform, as described in Settings..

Settings

Setting	Description
0	(Default) Right To Left
1	Left To Right
2	Top To Bottom
3	Bottom To Top

Remarks

This property determines which direction the caption text will begin scrolling. Scrolling must first be enabled by setting the **CaptionStyle** property to Scrolling Marquee, Bouncing Marquee or Sliding Marquee.

You can use the **MarqueeCycleBegin** event to change the direction of the scrolling once the control is displayed.

Note that when 0 or 1 is selected, the caption area is set to the maximum possible width based on the size of the control. A setting of 2 or 3 sets the caption area to the maximum possible height.

MarqueeScrollAmount Property

[See Also](#)

[Applies To](#)

Description

Returns or sets the number of pixels the marquee caption will be scrolled after each delay.

Syntax

object.**MarqueeScrollAmount** [= *number*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the number of pixels to scroll the marquee caption..

Remarks

The minimum setting for this property is 1.

This property controls the smoothness of the text scrolling effect. A smaller value will produce a smoother scroll, but the overall time it takes the text to scroll will be longer. A larger value results in faster but more choppy scrolling.

Use this property in conjunction with the **MarqueeScrollDelay** property to achieve the desired scrolling speed and smoothness.

MarqueeStyle Property

[See Also](#)

[Example](#)

[Applies To](#)

Description

Returns or sets the type of marquee effect the caption of the control will have.

Syntax

object.**MarqueeStyle** [= *number*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the type of marquee scrolling to perform, as described in Settings..

Settings

Setting	Description
0	(Default) None
1	Scrolling Marquee. The caption displays as text that moves constantly in one direction.
2	Sliding Marquee. The caption displays as text that moves initially then stops.
3	Blinking Marquee. The caption displays as text that blinks on and off.
4	Bouncing Marquee. The caption displays as text that moves constantly and reverses direction.

Remarks

ActiveThread controls now include a series of active caption styles. You can use these styles to give added impact to your application, or to call attention to particular areas of interest within your program.. Caption text is displayed in the *caption area* of the control; the size of this area is generally the size of the control, but may vary based on the specific control being used and on whether or not caption pictures are being displayed.

The movement of the caption is controlled by the **MarqueeDelay** and **MarqueeScrollAmount** properties. These two properties affect the speed and the smoothness of the scrolling, respectively.

When **MarqueeStyle** is set to 0, the caption will be static. Setting **MarqueeStyle** from 1 to 4 produces a moving caption. The direction of the movement is controlled by the **MarqueeDirection** property. When set to '1 - Scrolling,' the caption text will move continuously in one direction. The beginning of the text re-appears as soon as the last of the text has left the caption area. Before the text appears, a **MarqueeCycleBegin** event is fired.

When **MarqueeStyle** is set to '2- Sliding,' the caption text will move until it reaches the edge of the caption area. It then stops moving and appear static. If the text is too long to fit completely within the caption area, the caption will continue to scroll until all of the text has been displayed, then it will stop.

Blinking caption text does not move, but flashes on and off. The rate of the flashing is controlled by the **MarqueeDelay** property. The **MarqueeCycleBegin** event is fired just before the text appears.

A setting of '4- Bouncing' causes the text to move in 2 directions. The text will move in one direction until it reaches the edge of the caption area, when it will reverse direction. If the

caption text is too long to fit completely within the area, the caption will continue to scroll in one direction until all the text has been displayed, then it will reverse direction.

When **MarqueeStyle** is set to '4- Sliding,' the caption text will move until all of the text has been displayed in the caption area. It then stops moving and appears static.

MinHeight Property

[See Also](#)

[Applies To](#)

Description

Returns or sets the minimum possible height of the pane.

Syntax

object.**MinHeight** [= *number*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	A single precision expression specifying the minimum height of the pane, in units determined by the scale mode of the container.

Remarks

You can use this property to ensure that a particular pane is never resized below a certain height.

Note that this limitation applies only to resizing the pane using the splitter bars. You may still set the pane to any size through code, regardless of the value of this property.

This property takes effect at run time. At design time you can move the splitter bars without restriction.

MinWidth Property

[See Also](#)

[Applies To](#)

Description

Returns or sets the minimum possible width of the pane.

Syntax

object.**MinWidth** [= *number*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	A single precision expression specifying the minimum width of the pane, in units determined by the scale mode of the container.

Remarks

You can use this property to ensure that a particular pane is never resized below a certain width.

Note that this limitation applies only to resizing the pane using the splitter bars. You may still set the pane to any size through code, regardless of the value of this property.

This property takes effect at run time. At design time you can move the splitter bars without restriction.

MouseDown Event

[See Also](#)

[Applies To](#)

Description

Occurs when the user presses a mouse button while the mouse pointer is within the boundary of the control.

Syntax

Sub *control_MouseDown* ([*index* **As Integer**] *button* **As Integer**, *shift* **As Integer**, *x* **As Single**, *y* **As Single**)

The event parameters are:

Parameter Description

<i>index</i>	An integer expression that uniquely identifies a control if it is in a control array.
<i>button</i>	An integer expression that identifies the button that was pressed to cause the event. The button argument is a bit field with bits corresponding to the left button (bit 0), right button (bit 1), and middle button (bit 2). These bits correspond to the values 1, 2, and 4, respectively. Only one of the bits is set, indicating the button that caused the event.
<i>shift</i>	An integer expression that corresponds to the state of the SHIFT, CTRL, and ALT keys when the button specified in the button argument is pressed. A bit is set if the key is down.
<i>x,y</i>	A single-precision value that specifies the current location of the mouse pointer. The x and y values are always expressed in terms of the coordinate system set by the scale mode of the object's container.

Remarks

The *shift* argument is a bit field with the least-significant bits corresponding to the SHIFT key (bit 0), the CTRL key (bit 1), and the ALT key (bit 2). These bits correspond to the values 1, 2, and 4, respectively. The *shift* argument indicates the state of these keys. Some, all, or none of the bits can be set, indicating that some, all, or none of the keys are pressed. For example, if both CTRL and ALT are pressed, the value of *shift* is 6.

Unlike the **Click** and **DbClick** events, **MouseDown** and **MouseUp** events enable you to distinguish between the left, right, and middle mouse buttons. You can also write code for mouse-keyboard combinations that use the SHIFT, CTRL, and ALT keyboard modifiers.

Your development environment may have pre-defined constants for the values of *button* and *shift*; you should use them wherever possible.

MouseEnter Event

[See Also](#)

[Example](#)

[Applies To](#)

Description

Occurs when the mouse pointer enters the boundary of the control.

Syntax

Sub *control*_**MouseEnter** ([*index* **As Integer**] *button* **As Integer**, *shift* **As Integer**, *x* **As Single**, *y* **As Single**)

The event parameters are:

Parameter Description

<i>index</i>	An integer expression that uniquely identifies a control if it is in a control array.
<i>button</i>	An integer expression that identifies the button that was pressed when the event occurred. The button argument is a bit field with bits corresponding to the left button (bit 0), right button (bit 1), and middle button (bit 2). These bits correspond to the values 1, 2, and 4, respectively. Only one of the bits is set, indicating the button that caused the event.
<i>shift</i>	An integer expression that corresponds to the state of the SHIFT, CTRL, and ALT keys when the event occurred. A bit is set if the key is down.
<i>x,y</i>	A single-precision value that specifies the location of the mouse pointer when it entered the control. The x and y values are always expressed in terms of the coordinate system set by the scale mode of the object's container.

Remarks

This event is fired whenever the mouse pointer passes within the borders of the control. As the mouse pointer leaves the control, a **MouseExit** event is fired.

The *shift* argument is a bit field with the least-significant bits corresponding to the SHIFT key (bit 0), the CTRL key (bit 1), and the ALT key (bit 2). These bits correspond to the values 1, 2, and 4, respectively. The *shift* argument indicates the state of these keys. Some, all, or none of the bits can be set, indicating that some, all, or none of the keys are pressed. For example, if both CTRL and ALT are pressed, the value of *shift* is 6.

Unlike the **Click** and **DbClick** events, **MouseDown** and **MouseUp** events enable you to distinguish between the left, right, and middle mouse buttons. You can also write code for mouse-keyboard combinations that use the SHIFT, CTRL, and ALT keyboard modifiers.

Your development environment may have pre-defined constants for the values of *button* and *shift*; you should use them wherever possible.

MouseExit Event

[See Also](#)

[Example](#)

[Applies To](#)

Description

Occurs when the mouse pointer exits the boundary of the control

Syntax

Sub *control_MouseExit* (*[index As Integer]* *button As Integer*, *shift As Integer*, *x As Single*, *y As Single*)

The event parameters are:

Parameter Description

<i>index</i>	An integer expression that uniquely identifies a control if it is in a control array.
<i>button</i>	An integer expression that identifies the button that was pressed when the event occurred. The button argument is a bit field with bits corresponding to the left button (bit 0), right button (bit 1), and middle button (bit 2). These bits correspond to the values 1, 2, and 4, respectively. Only one of the bits is set, indicating the button that caused the event.
<i>shift</i>	An integer expression that corresponds to the state of the SHIFT, CTRL, and ALT keys when the event occurred. A bit is set if the key is down.
<i>x,y</i>	A single-precision value that specifies the location of the mouse pointer when it entered the control. The x and y values are always expressed in terms of the coordinate system set by the scale mode of the object's container.

Remarks

This event is fired whenever the mouse pointer was over the control and then moves outside its borders. When the mouse pointer first enters the control, a **MouseEnter** event is fired.

The *shift* argument is a bit field with the least-significant bits corresponding to the SHIFT key (bit 0), the CTRL key (bit 1), and the ALT key (bit 2). These bits correspond to the values 1, 2, and 4, respectively. The *shift* argument indicates the state of these keys. Some, all, or none of the bits can be set, indicating that some, all, or none of the keys are pressed. For example, if both CTRL and ALT are pressed, the value of *shift* is 6.

Unlike the **Click** and **DbClick** events, **MouseDown** and **MouseUp** events enable you to distinguish between the left, right, and middle mouse buttons. You can also write code for mouse-keyboard combinations that use the SHIFT, CTRL, and ALT keyboard modifiers.

Your development environment may have pre-defined constants for the values of *button* and *shift*; you should use them wherever possible.

MouseIcon Property

[See Also](#)

[Applies To](#)

Description

Returns or sets the icon that will be used for the mouse pointer when the **MousePointer** property is set to Custom.

Syntax

object.**MouseIcon** [= *picture*]

object.**MouseIcon** = **LoadPicture**(*pathname*)

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>picture</i>	An object expression that evaluates to a Picture, most commonly the Picture property from a Form object, PictureBox control, or Image control.
<i>pathname</i>	A string expression specifying the path and filename of the file containing the custom icon.

Remarks

The **MouseIcon** property provides a custom icon that is used when the **MousePointer** property is set to 99. You can use the **MouseIcon** property to load either cursor (.CUR) or icon (.ICO) files.

MouseMove Event

[See Also](#)

[Example](#)

[Applies To](#)

Description

Occurs when the mouse pointer is moved while within the boundaries of the control.

Syntax

Sub *control*_**MouseMove** ([*index* **As Integer**] *button* **As Integer**, *shift* **As Integer**, *x* **As Single**, *y* **As Single**)

The event parameters are:

Parameter Description

<i>index</i>	An integer expression that uniquely identifies a control if it is in a control array.
<i>button</i>	An integer expression that identifies the button that was pressed when the event occurred. The button argument is a bit field with bits corresponding to the left button (bit 0), right button (bit 1), and middle button (bit 2). These bits correspond to the values 1, 2, and 4, respectively. Only one of the bits is set, indicating the button that caused the event.
<i>shift</i>	An integer expression that corresponds to the state of the SHIFT, CTRL, and ALT keys when the event occurred. A bit is set if the key is down.
<i>x,y</i>	A single-precision value that specifies the location of the mouse pointer. The <i>x</i> and <i>y</i> values are always expressed in terms of the coordinate system set by the scale mode of the object's container.

Remarks

This event is fired whenever the mouse is moved within the borders of the control. As the mouse is moved within the boundaries of the control, this event will occur multiple times.

The *shift* argument is a bit field with the least-significant bits corresponding to the SHIFT key (bit 0), the CTRL key (bit 1), and the ALT key (bit 2). These bits correspond to the values 1, 2, and 4, respectively. The *shift* argument indicates the state of these keys. Some, all, or none of the bits can be set, indicating that some, all, or none of the keys are pressed. For example, if both CTRL and ALT are pressed, the value of *shift* is 6.

Unlike the **Click** and **DbClick** events, **MouseDown** and **MouseUp** events enable you to distinguish between the left, right, and middle mouse buttons. You can also write code for mouse-keyboard combinations that use the SHIFT, CTRL, and ALT keyboard modifiers.

Your development environment may have pre-defined constants for the values of *button* and *shift*; you should use them wherever possible.

MousePointer Property

[See Also](#)

[Applies To](#)

Description

Returns or sets a value specifying the type of mouse pointer displayed when the mouse is over a particular part of an object at run time.

Syntax

object.**MousePointer** [= *number*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the type of mouse pointer displayed, as described in Settings.

Settings

Setting	Description
0	(Default) Shape determined by the object.
1	Arrow.
2	Cross (cross-hair pointer).
3	I Beam.
4	Icon (small square within a square).
5	Size (four-pointed arrow pointing north, south, east, and west).
6	Size NE SW (double arrow pointing northeast and southwest).
7	Size N S (double arrow pointing north and south).
8	Size NW, SE (double arrow pointing northwest and southeast).
9	Size WE (double arrow pointing west and east).
10	Up Arrow.
11	Hourglass (wait).
12	No Drop.
13	Arrow and hourglass.
14	Arrow and question mark.
15	Size all.
99	Custom icon specified by the Mouselcon property.

Remarks

You can use this property when you want to indicate changes in functionality as the mouse pointer passes over controls on a form or dialog box. The Hourglass setting (11) is useful for indicating that the user should wait for a process or operation to finish.

MouseUp Event

[See Also](#)

[Applies To](#)

Description

Occurs when the user releases a mouse button while the mouse pointer is within the boundaries of the control.

Syntax

Sub *control_MouseUp* ([*index* **As Integer**] *button* **As Integer**, *shift* **As Integer**, *x* **As Single**, *y* **As Single**)

The event parameters are:

Parameter Description

<i>index</i>	An integer expression that uniquely identifies a control if it is in a control array.
<i>button</i>	An integer expression that identifies the button that was released to cause the event. The button argument is a bit field with bits corresponding to the left button (bit 0), right button (bit 1), and middle button (bit 2). These bits correspond to the values 1, 2, and 4, respectively. Only one of the bits is set, indicating the button that caused the event.
<i>shift</i>	An integer expression that corresponds to the state of the SHIFT, CTRL, and ALT keys when the button specified in the button argument is released. A bit is set if the key is down.
<i>x,y</i>	A single-precision value that specifies the current location of the mouse pointer. The x and y values are always expressed in terms of the coordinate system set by the scale mode of the object.

Remarks

The *shift* argument is a bit field with the least-significant bits corresponding to the SHIFT key (bit 0), the CTRL key (bit 1), and the ALT key (bit 2). These bits correspond to the values 1, 2, and 4, respectively. The *shift* argument indicates the state of these keys. Some, all, or none of the bits can be set, indicating that some, all, or none of the keys are pressed. For example, if both CTRL and ALT are pressed, the value of *shift* is 6.

Unlike the **Click** and **DbClick** events, **MouseDown** and **MouseUp** events enable you to distinguish between the left, right, and middle mouse buttons. You can also write code for mouse-keyboard combinations that use the SHIFT, CTRL, and ALT keyboard modifiers.

Your development environment may have pre-defined constants for the values of *button* and *shift*; you should use them wherever possible.

Name Property

Applies To

Description

Returns or sets a unique ID string for the specified object.

Syntax

object.**Name** [= *text*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>text</i>	A string expression that specifies a unique identifier for the object.

Remarks

This is an extender property, except when applied to the **Pane** or **Font** object.

The **Name** property is read-only at run time.

The default name for new objects is the kind of object plus a unique integer. For example, the first new `SSCommand` object is `SSCommand1`, and the second `SSPanel` control you create on a form is `SSPanel2`.

An object's `Name` property must start with a letter and can be a maximum of 40 characters. It can include numbers and underline (`_`) characters but can't include punctuation or spaces.

Name Property (Font object)

[See Also](#)

[Applies To](#)

Description

Returns or sets the font name of the **Font** object.

Syntax

object.**Name** [= *font*]

The **Name** property syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>font</i>	A string expression specifying the name of the font to be used.

Remarks

The **Font** object is not directly available at design time. Instead you set the **Name** property through a control's **Font** property.

At run time, however, you can set **Name** directly by specifying its setting for the **Font** object.

Non-Distributable Files

Non-distributable files are required to support the product in a development environment. Under the terms of your license agreement, you cannot distribute these files with your application. These include the executable and support files for the design-time environment and the design-time support files for product components.

 The following files may **NOT** be distributed:

SSPPG.DLL	Support files for the property pages
*.HLP	Any help or documentation files included with
*.TXT	ActiveThread

OLE Drag and Drop Operations

See Also

ActiveThread controls comply with the ActiveX standard, which means they are capable of handling OLE drag and drop operations from other applications or even the Windows shell. OLE Drag and Drop support is provided via a number of methods and events that work together to support both sides of a drag-and-drop conversation; the object being dragged, or "source" and the object receiving the drag, or "target".

In an OLE drag and drop situation, there are 2 groups of events that occur; *Drag Side* events and *Drop Side* events. Together these two groups constitute all the communication that goes on between the drag and drop source and target. The execution of these events is as follows:

Note that there are constants for most of the values you need to use when implementing drag-and-drop. See [Constants](#) for more information.

Drag Side (source control)

To start a drag and drop operation, you call the **OLEDrag** method. This initiates the drag and drop and fires an **OLEStartDrag** event in the source control.

The **OLEStartDrag** event receives parameters that include an **ssDataObject** object. This is an object that serves as the container for whatever information is being transferred by the drag-and-drop operation. The **ssDataObject** object can be queried to provide information about what is being dragged and what data types are supported. You also receive an *Effects* parameter that tells it what types of operations are allowed. Possible effects are none, move, copy, and scroll.

As the mouse pointer with the dragging object moves, multiple **OLEGiveFeedback** events are fired. The event is passed an *Effect* parameter, as well as a boolean *DefaultCursors* parameter that determines whether to display default or custom mouse pointers as the mouse moves. This event is used to determine when the dragging object is over a suitable target.

Drop Side (target control)

When a dragging object enters the control that is a possible target, the **OLEDragOver** event for the target control is fired.

When the mouse button is released, an **OLEDragDrop** event is fired in the target control.

Drag Side (source control)

If the drag and drop target needs more information than was supplied with the object (in **OLEStartDrag**) it will cause the **OLESetData** event to be fired in the source control. The source control can use this event to continue the conversation by supplying additional data.

When the object is dropped, an **OLECompleteDrag** event is fired in the source control.

Note that to handle an object being dragged from Windows itself or another application, only the drop side of the conversation must be implemented in your program. Similarly, if your application will be acting only as a source of draggable objects, you will only need to implement the drag side of the operation.

See Also

ssDataObject object

OLECompleteDrag Event

[See Also](#)

[Applies To](#)

Description

Occurs when a source component is dropped onto a target component, informing the source component that a drag action was either performed or canceled.

Syntax

Sub *control_OLECompleteDrag* ([*index* As **Integer**] *effect* As **Integer**)

The event parameters are:

Parameter Description

<i>index</i>	An integer expression that uniquely identifies a control if it is in a control array.
<i>effect</i>	A long integer set by the source object identifying the action that has been performed, thus allowing the source to take appropriate action if the component was moved (such as the source deleting data if it is moved from one component to another). The possible values are listed in Settings.

Settings

The settings for *effect* are:

Value	Description
0	Drop target cannot accept the data, or the drop operation was canceled.
1	Drop results in a copy of data from the source to the target. The original data is unaltered by the drag operation.
2	Drop results in a link to the original data being created between drag source and drop target.

Remarks

The **OLECompleteDrag** event is the final event to be called in an OLE drag/drop operation. This event informs the source component of the action that was performed when the object was dropped onto the target component. The target sets this value through the effect parameter of the **OLEDragDrop** event. Based on this, the source can then determine the appropriate action it needs to take. For example, if the object was moved into the target, the source needs to delete the object from itself after the move.

OLEDrag Method

[See Also](#)

[Applies To](#)

Description

This method causes the control to enter OLE Drag mode. OLE Drag mode ends when mouse button is released.

Syntax

object.**OLEDrag**

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list..

Remarks

When the OLEDrag method is called, the component's **OLEStartDrag** event occurs, allowing it to supply data to a target component.

OLEDragDrop Event

[See Also](#)

[Applies To](#)

Description

Occurs when a source component is dropped onto a target component when the source component determines that a drop can occur.

Syntax

Sub *control_OLEDragDrop* ([*index As Integer*] *data As ssDataObject*, *effect As Integer*, *button As Integer*, *shift As Integer*, *x As Single*, *y As Single*)

The event parameters are:

Parameter Description

<i>index</i>	An integer expression that uniquely identifies a control if it is in a control array.
<i>data</i>	A ssDataObject object containing formats that the source will provide and, in addition, possibly the data for those formats. If no data is contained in the ssDataObject , it is provided when the control calls the OLEGetData method.
<i>effect</i>	A long integer set by the target component identifying the action that has been performed (if any), thus allowing the source to take appropriate action if the component was moved (such as the source deleting the data). The possible values are listed in Settings.
<i>button</i>	An integer expression that identifies the button that was released to cause the event. The button argument is a bit field with bits corresponding to the left button (bit 0), right button (bit 1), and middle button (bit 2). These bits correspond to the values 1, 2, and 4, respectively. Only one of the bits is set, indicating the button that caused the event.
<i>shift</i>	An integer expression that corresponds to the state of the SHIFT, CTRL, and ALT keys when the button specified in the button argument is released. A bit is set if the key is down.
<i>x,y</i>	A single-precision value that specifies the current location of the mouse pointer. The x and y values are always expressed in terms of the coordinate system set by the scale mode of the object.

Settings

The settings for *effect* are:

Value	Description
0	Drop target cannot accept the data.
1	Drop results in a copy of data from the source to the target. The original data is unaltered by the drag operation.
2	Drop results in data being moved from drag source to drop source. The drag source should remove the data from itself after the move.

Remarks

The source ActiveX component should always mask values from the *effect* parameter to ensure compatibility with future implementations of ActiveX components. Presently, only three of the 32 bits in the effect parameter are used. In the future, these other bits may be used. Therefore, as a precaution against future problems, drag sources and drop targets should mask these values appropriately before performing any comparisons.

For example, a source component should not compare an effect against, say, the number

1, such as in this manner:

```
If Effect = 1 Then ...
```

Instead, the source component should mask for the value or values being sought, such as this:

```
If Effect And 1 = 1 Then ...
```

This allows for the definition of new drop effects in the future while preserving backwards compatibility with your existing code.

The *shift* argument is a bit field with the least-significant bits corresponding to the SHIFT key (bit 0), the CTRL key (bit 1), and the ALT key (bit 2). These bits correspond to the values 1, 2, and 4, respectively. The *shift* argument indicates the state of these keys. Some, all, or none of the bits can be set, indicating that some, all, or none of the keys are pressed. For example, if both CTRL and ALT are pressed, the value of *shift* is 6.

Unlike the **Click** and **DbClick** events, **MouseDown** and **MouseUp** events enable you to distinguish between the left, right, and middle mouse buttons. You can also write code for mouse-keyboard combinations that use the SHIFT, CTRL, and ALT keyboard modifiers.

Your development environment may have pre-defined constants for the values of *button* and *shift*; you should use them wherever possible.

OLEDragOver Event

[See Also](#) [Applies To](#)

Description

Occurs when one component is dragged over another.

Syntax

Sub *control_OLEDragOver* ([*index* As **Integer**] *data* As **ssDataObject**, *effect* As **Integer**, *button* As **Integer**, *shift* As **Integer**, *x* As **Single**, *y* As **Single**)

The event parameters are:

Parameter Description

<i>index</i>	An integer expression that uniquely identifies a control if it is in a control array.
<i>data</i>	A ssDataObject object containing formats that the source will provide and, in addition, possibly the data for those formats. If no data is contained in the ssDataObject , it is provided when the control calls the OLEGetData method.
<i>effect</i>	A long integer set by the target component identifying the action that has been performed (if any), thus allowing the source to take appropriate action if the component was moved (such as the source deleting the data). The possible values are listed in Settings.
<i>button</i>	An integer expression that identifies the button that was released to cause the event. The button argument is a bit field with bits corresponding to the left button (bit 0), right button (bit 1), and middle button (bit 2). These bits correspond to the values 1, 2, and 4, respectively. Only one of the bits is set, indicating the button that caused the event.
<i>shift</i>	An integer expression that corresponds to the state of the SHIFT, CTRL, and ALT keys when the button specified in the button argument is released. A bit is set if the key is down.
<i>x,y</i>	A single-precision value that specifies the current location of the mouse pointer. The x and y values are always expressed in terms of the coordinate system set by the scale mode of the object.
<i>state</i>	An integer that corresponds to the transition state of the control being dragged in relation to a target form or control. The possible values are listed in Settings.

Settings

The settings for *effect* are:

Value	Description
0	Drop target cannot accept the data.
1	Drop results in a copy of data from the source to the target. The original data is unaltered by the drag operation.
2	Drop results in data being moved from drag source to drop source. The drag source should remove the data from itself after the move.
-2147483648	(&H80000000) Scrolling is occurring or about to occur in the target component. This value is used in conjunction with the other values. Use only if you are performing your own scrolling in the target component.

The settings for *state* are:

Value	Description
0	Source component is being dragged within the range of a target.
1	Source component is being dragged out of the range of a target.
2	Source component has moved from one position in the target to another.

Remarks

Note If the value of the *state* parameter is 1, indicating that the mouse pointer has left the target, then the *x* and *y* parameters will contain zeros.

The source ActiveX component should always mask values from the *effect* parameter to ensure compatibility with future implementations of ActiveX components. Presently, only three of the 32 bits in the effect parameter are used. In the future, these other bits may be used. Therefore, as a precaution against future problems, drag sources and drop targets should mask these values appropriately before performing any comparisons.

For example, a source component should not compare an effect against, say, the number 1, such as in this manner:

```
If Effect = 1 Then ...
```

Instead, the source component should mask for the value or values being sought, such as this:

```
If Effect And 1 = 1 Then ...
```

This allows for the definition of new drop effects in the future while preserving backwards compatibility with your existing code.

The *shift* argument is a bit field with the least-significant bits corresponding to the SHIFT key (bit 0), the CTRL key (bit 1), and the ALT key (bit 2). These bits correspond to the values 1, 2, and 4, respectively. The *shift* argument indicates the state of these keys. Some, all, or none of the bits can be set, indicating that some, all, or none of the keys are pressed. For example, if both CTRL and ALT are pressed, the value of *shift* is 6.

Unlike the **Click** and **DbClick** events, **MouseDown** and **MouseUp** events enable you to distinguish between the left, right, and middle mouse buttons. You can also write code for mouse-keyboard combinations that use the SHIFT, CTRL, and ALT keyboard modifiers.

Your development environment may have pre-defined constants for the values of *button* and *shift*; you should use them wherever possible.

OLEDropMode Property

[See Also](#)

[Applies To](#)

Description

Returns or sets a value that determines whether the control can be a drop target for OLE drag-and-drop operations.

Syntax

object.**OLEDropMode** [= *number*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying how the control will handle OLE drag-and-drop operations, as described in Settings..

Settings

Setting	Description
0	The control will not support OLE drag operations.
1	(Default) The control supports manual OLE drop mode.

Remarks

Note In Visual Basic, the target component inspects what is being dragged over it in order to determine which events to trigger; the OLE drag/drop events, or the Visual Basic drag/drop events. There is no collision of components or confusion about which events are fired, since only one type of object can be dragged at a time.

OLEGiveFeedback Event

[See Also](#)

[Applies To](#)

Description

Occurs after every **OLEDragOver** event. **OLEGiveFeedback** allows the source component to provide visual feedback to the user, such as changing the mouse cursor to indicate what will happen if the user drops the object, or provide visual feedback on the selection (in the source component) to indicate what will happen.

Syntax

Sub *control_***OLEGiveFeedback** ([*index* **As Integer**] *effect* **As Long**, *defaultcursors* **As Boolean**)

The event parameters are:

Parameter	Description
<i>index</i>	An integer expression that uniquely identifies a control if it is in a control array.
<i>effect</i>	A long integer set by the target component in the OLEDragOver event specifying the action to be performed if the user drops the selection on it. This allows the source to take the appropriate action (such as giving visual feedback). The possible values are listed in Settings.
<i>defaultcursors</i>	A boolean value which determines whether Visual Basic uses the default mouse cursor proved by the component, or uses a user-defined mouse cursor, as described in Settings.

Settings

The settings for *effect* are:

Value	Description
0	Drop target cannot accept the data.
1	Drop results in a copy of data from the source to the target. The original data is unaltered by the drag operation.
2	Drop results in data being moved from drag source to drop source. The drag source should remove the data from itself after the move.
-2147483648 (&H80000000)	Scrolling is occurring or about to occur in the target component. This value is used in conjunction with the other values. Note Use only if you are performing your own scrolling in the target component.

The settings for *defaultcursors* are:

Value	Description
True	(Default) Default mouse cursor is used.
False	Use a custom mouse cursor.

Remarks

If there is no code in the **OLEGiveFeedback** event, or if the *defaultcursors* parameter is set to True, then the mouse cursor is automatically set to the default cursor provided by the component.

The source ActiveX component should always mask values from the *effect* parameter to ensure compatibility with future implementations of ActiveX components. Presently, only three of the 32 bits in the effect parameter are used. In the future, these other bits may be

used. Therefore, as a precaution against future problems, drag sources and drop targets should mask these values appropriately before performing any comparisons.

For example, a source component should not compare an effect against, say, the number 1, such as in this manner:

```
If Effect = 1 Then ...
```

Instead, the source component should mask for the value or values being sought, such as this:

```
If Effect And 1 = 1 Then ...
```

This allows for the definition of new drop effects in the future while preserving backwards compatibility with your existing code.

The *shift* argument is a bit field with the least-significant bits corresponding to the SHIFT key (bit 0), the CTRL key (bit 1), and the ALT key (bit 2). These bits correspond to the values 1, 2, and 4, respectively. The *shift* argument indicates the state of these keys. Some, all, or none of the bits can be set, indicating that some, all, or none of the keys are pressed. For example, if both CTRL and ALT are pressed, the value of *shift* is 6.

Unlike the **Click** and **DbClick** events, **MouseDown** and **MouseUp** events enable you to distinguish between the left, right, and middle mouse buttons. You can also write code for mouse-keyboard combinations that use the SHIFT, CTRL, and ALT keyboard modifiers.

Your development environment may have pre-defined constants for the values of *button* and *shift*; you should use them wherever possible.

OLESetData Event

[See Also](#)

[Applies To](#)

Description

Occurs on a source component when a target component performs the GetData method on the source's **ssDataObject** object, but the data for the specified format has not yet been loaded.

Syntax

Sub *control_OLESetData* (*[index As Integer]* *data* As **ssDataObject**, *dataformat* As **Integer**)

The event parameters are:

Parameter Description

<i>index</i>	An integer expression that uniquely identifies a control if it is in a control array.
<i>data</i>	An ssDataObject object in which to place the requested data. The component calls the SetData method to load the requested format.
<i>dataformat</i>	An integer expression specifying the format of the data that the target component is requesting. The source component uses this value to determine what to load into the ssDataObject object.

Settings

The settings for *dataformat* are:

Setting	Description
1	Text (.TXT files)
2	Bitmap (.BMP files)
3	metafile (.WMF files)
8	Device-independent bitmap (DIB)
9	Color palette
14	Enhanced metafile (.EMF files)
15	List of files
-16639	Rich text format (.RTF files)

Remarks

In certain cases, you may wish to defer loading data into the **ssDataObject** object of a source component to save time, especially if the source component supports many formats. This event allows the source to respond to only one request for a given format of data. When this event is called, the source should check the format parameter to determine what needs to be loaded and then perform the **SetData** method on the **ssDataObject** object to load the data which is then passed back to the target component.

Constants that correspond to these formats are available. See [Constants](#) for more information.

OLEStartDrag Event

[See Also](#)

[Applies To](#)

Description

Occurs when a component's OLEDrag method is performed. This event specifies the data formats and drop effects that the source component supports. It can also be used to insert data into the **ssDataObject** object.

Syntax

Sub *control_OLEStartDrag* ([*index As Integer*] *data As ssDataObject*, *allowedeffects As Long*)

The event parameters are:

Parameter	Description
<i>index</i>	An integer expression that uniquely identifies a control if it is in a control array.
<i>data</i>	An ssDataObject object containing formats that the source will provide and, optionally, the data for those formats. If no data is contained in the ssDataObject , it is provided when the control calls the GetData method. The programmer should provide the values for this parameter in this event.
<i>allowedeffects</i>	A long integer containing the effects that the source component supports. The possible values are listed in Settings. The programmer should provide the values for this parameter in this event.

Settings

The settings for *allowedeffects* are:

Value	Description
0	Drop target cannot accept the data.
1	Drop results in a copy of data from the source to the target. The original data is unaltered by the drag operation.
2	Drop results in data being moved from drag source to drop source. The drag source should remove the data from itself after the move.

Remarks

The source component should logically OR together the supported values and place the result in the *allowedeffects* parameter. The target component can use this value to determine the appropriate action (and what the appropriate user feedback should be).

You may wish to defer putting data into the **ssDataObject** object until the target component requests it. This allows the source component to save time by not loading multiple data formats.

When the target performs the **GetData** method on the **ssDataObject** object, the source's **OLESetData** event will occur if the requested data is not contained in the **ssDataObject**. At this point, the data can be loaded into the **ssDataObject**, which will in turn provide the data to the target.

If the user does not load any formats into the **ssDataObject**, then the drag/drop operation is canceled.

OptionBtnGraphics Property

[See Also](#)

[Example](#)

[Applies To](#)

Description

Returns or sets the image to be used for the option button portion of the control.

Syntax


object.**OptionBtnGraphics** [= *picture*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>picture</i>	A picture object specifying a graphic, as described in Settings.

Settings

Setting	Description
(None)	(Default) No picture.
(Bitmap)	Specifies a graphic. You can load the graphic from the properties window at design time. At run time, you can set this property using the LoadPicture function on a bitmap.

Remarks

You can specify a custom appearance for the button portion of the control using this property, which will replace the standard option button graphic . This is useful for designing a distinctive user interface.

The picture specified for this property is a segmented bitmap containing all the possible states of the control. Several sample bitmaps are included with ActiveThread to get you started in designing your own option button multiple-state bitmaps. The SSOption control has six states. If you are creating custom graphics, your bitmap should have six segments of equal width. These six states should appear from left to right in the segmented bitmap. The states of the control are:

0. **Unselected** - the normal state of the control when unchecked
1. **Selected** - the normal state of the control when checked
2. **Unselected Pressed** - unchecked, with the left mouse button being pressed while over the control
3. **Selected Pressed** - checked, with the left mouse button being pressed while over the control
4. **Unselected Disabled** - the disabled state of the control when unchecked
5. **Selected Disabled** - the disabled state of the control when checked

You can specify part of the option button graphic as transparent, using the **OptionBtnMaskColor** and **OptionBtnUseMask** properties.

OptionBtnMaskColor Property

[See Also](#)

[Example](#)

[Applies To](#)

Description

Returns or sets the color that will become the transparent part of the option button graphic.

Syntax

object. **OptionBtnMaskColor** [= *color*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>color</i>	A value or constant that determines the mask color of the option button graphic.

Remarks

This setting only takes effect when **OptionBtnUseMask** is set to True. This property works in conjunction with the **OptionBtnGraphics** property to create a custom graphic for the check box that replaces the standard check box graphic .

By specifying one of the colors used in the segmented bitmap as the **OptionBtnMaskColor**, you cause that color to become transparent when used by the control. This gives you the ability to design check box graphics that have shapes other than square, or that have transparent areas. This is especially important when designing option button graphics, since option buttons are generally round.

OptionBtnUseMask Property

[See Also](#)

[Example](#)

[Applies To](#)

Description

Returns or sets a value that determines whether the control will use the **OptionBtnMaskColor** to create transparent areas.

Syntax

object. **OptionBtnUseMask** [= *boolean*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying whether the mask color will be used, as described in Settings.

Settings

Setting	Description
True	The mask color will be used.
False	(Default) The mask color will not be used.

Remarks

This property enables the use of a mask color in the segmented bitmap specified by the **OptionBtnGraphics** property. When set to True, the color specified by the **OptionBtnMaskColor** property will become transparent in the control. When set to False, all colors in the segmented bitmap will be opaque and visible.

For more information on creating custom check box graphics, see the **OptionBtnGraphics** property.

Outline Property

Applies To

Description

Returns or sets a value that determines whether the control is displayed with a 1-pixel black border around its outer edge.

Syntax

object.**Outline** [= *boolean*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying the visibility of the control's border, as described in Settings.

Settings

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

Remarks

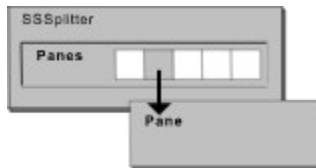
This property only applies to controls with Windows 3.X style borders. For other border settings this property has no effect.

Pane Object

[See Also](#)

[Example](#)

[Applies To](#)



Description

A **Pane** object is an area of the Splitter control, bounded either by splitter bars, the borders of the control or both. Each **Pane** object is a container which may hold one child control.

A **Pane** object is always a member of the [Panes collection](#) of the control.

Properties

[Control](#)

[ControlName](#)

[Height](#)

[Left](#)

[LockHeight](#)

[LockWidth](#)

[MinHeight](#)

[MinWidth](#)

[Name](#)

[Top](#)

[Width](#)

PaneFromControl Method

[See Also](#)

[Example](#)

[Applies To](#)

Description

This method returns the **Pane** object associated with a particular control.

Syntax

object.**PaneFromControl** ("*control*")

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list..
<i>control</i>	A child control of the Splitter control for which you wish to find the corresponding pane or window handle.

Remarks

Use this method when you wish to find out which pane contains a particular control.

PaneFromPosition Method

[See Also](#)

[Example](#)

[Applies To](#)

Description

This method returns the **Pane** object associated with a particular set of coordinates.

Syntax

object.**PaneFromPosition** (x, y)

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list..
<i>x, y</i>	A single-precision value that specifies a point somewhere within the boundaries of the control. The x and y values are always expressed in terms of the coordinate system set by the scale mode of the object.

Remarks

Use this method when you want to find out which pane is at a specific position. The most common use for this method would be to determine which pane is underneath the mouse pointer at a given time.

Use this method when you are reasonably sure that the coordinates you are passing are within the boundaries of a pane. If there is no pane located at the specified coordinates (i.e. the point is on a splitter bar, on the border of the control, or outside the control's client area) a trappable error will occur.

You can also use the **PaneFromPositionEx** method, which functions identically to the **PaneFromPosition** method except that it does not produce an error if the coordinates are not inside a pane.

PaneFromPositionEx Method

[See Also](#)

[Example](#)

[Applies To](#)

Description

This method returns the **Pane** object associated with a particular set of coordinates.

Syntax

object.**PaneFromPositionEx** (*x*, *y*)

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list..
<i>x</i> , <i>y</i>	A single-precision value that specifies a point somewhere within the boundaries of the control. The <i>x</i> and <i>y</i> values are always expressed in terms of the coordinate system set by the scale mode of the object.

Remarks

Use this method when you want to find out which pane is at a specific position. The most common use for this method would be to determine which pane is underneath the mouse pointer at a given time.

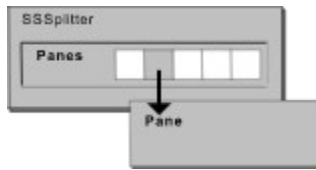
This method does not raise an error if the specified point is not located inside a pane. If the point is not within a pane, the method returns a value of **Nothing**.

Panes Collection

[See Also](#)

[Example](#)

[Applies To](#)



Description

The **Panes** collection is a collection of all the **Pane** objects in a Splitter control.

Properties

[Count](#)

Methods

[Add](#)

[Item](#)

[Remove](#)

Picture Property

[See Also](#)

[Applies To](#)

Description

Returns or sets a **Picture** object for display.

Syntax

object.**Picture** [= *picture*]

Part	Description
-------------	--------------------

<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
---------------	---

<i>picture</i>	A picture object specifying a graphic, as described in Settings.
----------------	--

Settings

Setting	Description
----------------	--------------------

(None)	(Default) No picture.
--------	-----------------------

(Bitmap, icon, metafile)	Specifies a graphic. You can load the graphic from the properties window at design time. At run time, you can set this property using the LoadPicture function on a bitmap, icon or metafile.
--------------------------	--

Remarks

This property will set the picture for the specified object. Any object can have a bitmap for the **Picture** property; only certain objects will accept an icon for the **Picture** property.

A picture is displayed in the control based on the **PictureAlignment** property. The picture is associated with the control's caption and is separate from the background picture specified by the **PictureBackground** property.

The **Picture** property is also used to provide animation for the control. This is accomplished by specifying a segmented bitmap for the **Picture** property. Each segment becomes a single frame of the animation, and you specify the number of segments (frames) to use by setting the **PictureFrames** property.

For some controls, the **Picture** property is only one of two or more properties controlling the display of pictures or animation. This is true of the **SSCommand** and **SSRibbon** controls, where separate pictures can be specified for the different states of the control.

PictureAlignment Property

[See Also](#)

[Applies To](#)

Description

Returns or sets a value that specifies how the picture will be aligned on the control.

Syntax

object.**PictureAlignment**[= *setting*]

Settings

The **PictureAlignment** property has the following settings for the **SSCheck**, **SSFrame** and **SSOption** controls:

Setting	Description
0	(Default) Left of Caption
1	Right of Caption
2	Justified (does not apply to SSFrame)

The **PictureAlignment** property has the following settings for the **SSCommand**, **SSPanel** and **SSRibbon** controls:

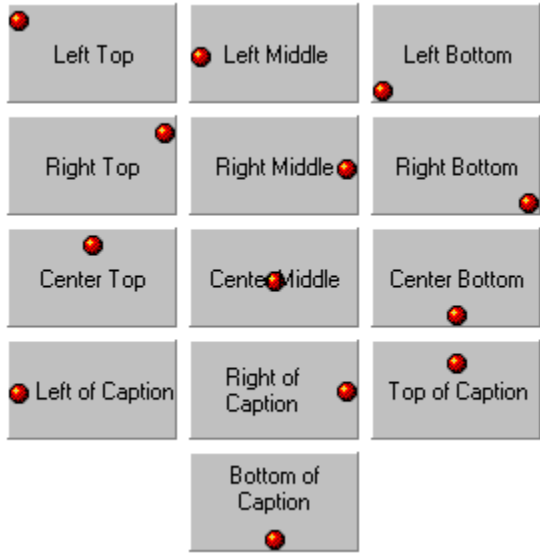
Setting	Description
0	(Default - SSPanel) Left Top
1	Left Middle
2	Left Bottom
3	Right Top
4	Right Middle
5	Right Bottom
6	Center Top
7	(Default - SSCommand & SSRibbon) Center Middle
8	Center Bottom
9	Left of Caption
10	Right of Caption
11	Top Of Caption
12	Bottom Of Caption

Remarks

The Justified setting of the SSCheck and SSOption controls aligns the picture to the edge of the control opposite the check box or option button. Therefore, this setting will be affected by the value of the **Alignment** property of the control.

Use the **PictureAlignment** property in conjunction with the **Alignment** property to produce a variety of design effects.

Example



PictureAnimationDelay Property

[See Also](#)

[Example](#)

[Applies To](#)

Description

Returns or sets the number of milliseconds between frames of an animation effect.

Syntax

object.**PictureAnimationDelay** [= *number*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the number of milliseconds between animation frames.

Remarks

Decreasing the number of milliseconds increases the smoothness of the animation, but causes the animation to require more system resources. A higher value for this setting produces more choppy animation.

Warning! Setting PictureAnimationDelay to 0 or a low number (less than 55 for Windows 95, less than 10 for Windows NT) will increase the demands the control places on the system's CPU. Depending on the machine running the application, this can have a significant effect on overall system performance.

PictureAnimationEnabled Property

[See Also](#)

[Example](#)

[Applies To](#)

Description

Returns or sets a value which determines whether the animation will play or not.

Syntax

object.**PictureAnimationEnabled** [= *boolean*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying whether to play the animation, as described in Settings.

Settings

Setting	Description
True	(Default) The animation will play.
False	The animation will remain a static picture with a single frame of the animation being displayed.

Remarks

Use this property to activate and deactivate the animation at run time, for example, you might enable the animation in response to a **MouseEnter** event, and disable it in the **MouseExit** event.

PictureBackground Property

[See Also](#)

[Applies To](#)

Description

Returns or sets a picture for display in the background of the control.

Syntax

object.**PictureBackground** [= *picture*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>picture</i>	A picture object specifying a graphic, as described in Settings.

Settings

Setting	Description
(None)	(Default) No picture.
(Bitmap, icon, metafile)	Specifies a graphic. You can load the graphic from the properties window at design time. At run time, you can set this property using the LoadPicture function on a bitmap, icon or metafile.

Remarks

The display of the background picture is controlled by the **BackStyle** property. When **BackStyle** is set to any picture style this is the picture used.

To display a picture on the surface of the control or in the caption of the control, use the **Picture** property.

Note **PictureBackground** is affected by the **PictureMaskColor** and **PictureUseMask** properties in the same way as the **Picture** property. You can therefore create background pictures with transparent areas. The color specified by the **PictureMaskColor** property will be transparent in both the caption picture (specified by the **Picture** property) and in the background picture (specified by the **PictureBackground** property.)

PictureBackgroundStyle Property

[See Also](#)

[Applies To](#)

Description

Returns or sets a value that determines whether the background picture of the control will be centered, stretched or tiled.

Syntax

object.**PictureBackgroundStyle** [= *integer*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>integer</i>	An integer expression specifying the type of background style to use, as described in Settings.

Settings

Setting	Description
0	Centered. The area of the control will be filled with a picture, as specified by the PictureBackground property. The picture will appear actual size in the center of the control and be clipped by the control's boundaries. If the picture does not fill the control's area, it will be bounded by an area that is either transparent or filled with the control's BackColor property, as determined by the BackStyle property..
1	Stretched. The area of the control will be filled with a picture, as specified by the PictureBackground property. The picture will be stretched or shrunk horizontally and/or vertically to fill the area of the control.
2	Tiled. The area of the control will be filled with a picture, as specified by the PictureBackground property. The picture will be tiled from the upper left hand corner of the control, and will be repeated as many times as necessary to fill the area of the control.

Remarks

The SSFrame and SSPanel controls now support several new background styles designed to make the controls integrate smoothly into applications with textured backgrounds.

The new **PictureBackgroundStyle** property enables you to set a different, complementary background for the Frame or Panel control. The Centered option displays a picture at actual size in the center of the control. If the picture is smaller than the control, it will be surrounded by a background region. If the picture is larger than the control, it will be cropped by the edges of the control. This setting is similar to the Centered setting available when specifying wallpaper for the Windows desktop.

The Stretched option will automatically alter the size of the picture to match that of the control, enlarging or shrinking it as necessary. The entire area of the control will be filled with the specified picture.

The Tiled option will repeat the specified picture, starting in the upper left corner of the control, as many times as needed to fill the area of the control. If the picture is larger than the control, it will be cropped by the right and bottom edges of the control. This setting is similar to the Tiled setting available when specifying wallpaper for the Windows desktop.

PictureDisabled Property

[See Also](#)

[Applies To](#)

Description

Returns or sets the image to be used for the button when it is disabled.

Syntax

object.**PictureDisabled** [= *picture*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>picture</i>	A picture object specifying a graphic, as described in Settings.

Settings

Setting	Description
(None)	(Default) No picture.
(Bitmap, icon, metafile)	Specifies a graphic. You can load the graphic from the properties window at design time. At run time, you can set this property using the LoadPicture function on a bitmap, icon or metafile.

Remarks

Command and Ribbon buttons can have different pictures for their up, down and disabled states. You set the picture for the up state using the standard **Picture** property. You can optionally specify a down and/or a disabled state by specifying a picture for, respectively, the **PictureDn** property and the **PictureDisabled** property.

If no pictures are specified for the additional button states, the picture specified by the **Picture** property will be used for all states of the button.

This property is also used to provide animation for the disabled state of the control. This is accomplished by specifying a segmented bitmap for the **PictureDisabled** property. Each segment becomes a single frame of the animation, and you specify the number of segments (frames) to use by setting the **PictureDisabledFrames** property.

PictureDisabledFrames Property

[See Also](#)

[Applies To](#)

Description

Returns or sets the number of segments (frames) that will be used when animating the control's picture when the control is disabled.

Syntax

object.**PictureDisabledFrames** [= *number*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the number of segments in the disabled picture.

Remarks

The SSCommand and SSRibbon controls give you the option to replace the default appearance of the control with a bitmap that you specify. You can also display animation on these controls by using a segmented bitmap. Each segment in the bitmap becomes a frame in the animation.

The property pages for the ActiveThreed controls give you an automated interface for importing individual bitmaps into your animation, which is then stored as a segmented bitmap. Additionally, you can specify any segmented bitmap you have previously created.

In either case, the overall width of the bitmap is divided evenly into the number of segments specified by the **PictureDisabledFrames** property.

PictureDn Property

[See Also](#)

[Applies To](#)

Description

Returns or sets the image to be used for the 'down' state of a button.

Syntax

object.**PictureDn** [= *picture*]

Part	Description
-------------	--------------------

<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
---------------	---

<i>picture</i>	A picture object specifying a graphic, as described in Settings.
----------------	--

Settings

Setting	Description
----------------	--------------------

(None)	(Default) No picture.
--------	-----------------------

(Bitmap, icon, metafile)	Specifies a graphic. You can load the graphic from the properties window at design time. At run time, you can set this property using the LoadPicture function on a bitmap, icon or metafile.
--------------------------	--

Remarks

Command and Ribbon buttons can have different pictures for their up, down and disabled states. You set the picture for the up state using the standard **Picture** property. You can optionally specify a down and/or a disabled state by specifying a picture for, respectively, the **PictureDn** property and the **PictureDisabled** property.

If no pictures are specified for the additional button states, the picture specified by the **Picture** property will be used for all states of the button.

This property is also used to provide animation for the down state of the control. This is accomplished by specifying a segmented bitmap for the **PictureDn** property. Each segment becomes a single frame of the animation, and you specify the number of segments (frames) to use by setting the **PictureDnFrames** property.

PictureDnChange Property

[See Also](#)

[Applies To](#)

Description

Returns or sets a value that specifies how the 'down' state of a button is created.

Syntax

object.**PictureDnChange** [= *number*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the method for creating the pressed-in state of the button, as described in Settings.

Settings

Setting	Description
0	(Default) No change. The picture will look the same in both states of the control.
1	Dither. The down state of the button will be created by dithering the picture used for the up state.
2	Invert. The down state of the button will be created by reversing the colors used for the up state.

Remarks

This property applies only when no picture has been specified in the **PictureDn** property.

PictureDnDisabled Property

[See Also](#)

[Applies To](#)

Description

Returns or sets the image to be used for the 'down' state of a button when it is disabled.

Syntax

object.**PictureDnDisabled** [= *picture*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>picture</i>	A picture object specifying a graphic, as described in Settings.

Settings

Setting	Description
(None)	(Default) No picture.
(Bitmap, icon, metafile)	Specifies a graphic. You can load the graphic from the properties window at design time. At run time, you can set this property using the LoadPicture function on a bitmap, icon or metafile.

Remarks

Ribbon buttons can have different pictures for their four different states: up, down, disabled and down disabled. You set the picture for the up state using the standard **Picture** property. You can optionally specify a down state by specifying a picture for the **PictureDn** property.

When the button is disabled, the **PictureDisabled** and the **PictureDnDisabled** properties are used to specify the pictures for the up and down states, respectively.

This property is also used to provide animation for the down disabled state of the control. This is accomplished by specifying a segmented bitmap for the **PictureDnDisabled** property. Each segment becomes a single frame of the animation, and you specify the number of segments (frames) to use by setting the **PictureDnDisabledFrames** property.

If no pictures are specified for the additional button states, the picture specified by the **Picture** property will be used for all states of the button.

PictureDnDisabledFrames Property

[See Also](#)

[Applies To](#)

Description

Returns or sets the number of segments (frames) that will be used when animating the control's picture when the control is 'down' and disabled.

Syntax

object.**PictureDisabledFrames** [= *number*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the number of segments in the disabled picture.

Remarks

The SSRibbon control gives you the option to replace the default appearance of the control with a bitmap that you specify. You can also display animation on this control by using a segmented bitmap. Each segment in the bitmap becomes a frame in the animation.

The property pages for the ActiveThreed controls give you an automated interface for importing individual bitmaps into your animation, which is then stored as a segmented bitmap. Additionally, you can specify any segmented bitmap you have previously created.

In either case, the overall width of the bitmap is divided evenly into the number of segments specified by the **PictureDisabledFrames** property.

PictureDnFrames Property

[See Also](#)

[Applies To](#)

Description

Returns or sets the number of segments (frames) that will be used when animating the control's picture when the control is 'down'.

Syntax

object.**PictureDnFrames** [= *number*]

Part	Description
-------------	--------------------

<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
---------------	---

<i>number</i>	An integer expression specifying the number of segments in the down picture.
---------------	--

Remarks

The SSCommand and SSRibbon controls give you the option to replace the default appearance of the control with a bitmap that you specify. You can also display animation on these controls by using a segmented bitmap. Each segment in the bitmap becomes a frame in the animation.

The property pages for the ActiveThreed controls give you an automated interface for importing individual bitmaps into your animation, which is then stored as a segmented bitmap. Additionally, you can specify any segmented bitmap you have previously created.

In either case, the overall width of the bitmap is divided evenly into the number of segments specified by the **PictureDnFrames** property.

PictureFrames Property

[See Also](#)

[Applies To](#)

Description

Returns or sets the number of segments (frames) that will be used when animating the control's picture.

Syntax

object.**PictureFrames** [= *number*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the number of segments in the picture.

Remarks

The SSCommand and SSRibbon controls give you the option to replace the default appearance of the control with a bitmap that you specify. You can also display animation on these controls by using a segmented bitmap. Each segment in the bitmap becomes a frame in the animation.

The property pages for the ActiveThreed controls give you an automated interface for importing individual bitmaps into your animation, which is then stored as a segmented bitmap. Additionally, you can specify any segmented bitmap you have previously created.

In either case, the overall width of the bitmap is divided evenly into the number of segments specified by the **PictureDisabledFrames** property.

PictureMaskColor Property

[See Also](#)

[Applies To](#)

Description

Returns or sets the color in a picture that will be made transparent.

Syntax

object.**PictureMaskColor** [= *color*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>color</i>	A long integer value or constant that determines the color which will become transparent.

Remarks

When you are designing a picture for use on a control you often want part of the picture to be transparent, particularly the area surrounding the outlines of the picture. This prevents a rectangle of a fixed color from appearing around the picture if there is a change to the background color of the control or the application.

You can achieve a transparent effect by specifying a *mask color* when setting up your application. The mask color tells the control which color in your picture will become transparent. The mask color for the caption picture (specified by the **Picture** property) is specified using the **PictureMaskColor** property.

PictureUseMask Property

[See Also](#)

[Applies To](#)

Description

Returns or sets a value that determines whether the picture will have a transparent color.

Syntax

object.**PictureUseMask** [= *boolean*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying whether the mask color will be used to create a transparent picture, as described in Settings.

Settings

Setting	Description
True	Mask color will be used. Picture will be transparent wherever mask color appears.
False	(Default) Mask color will not be used. Picture will be opaque.

Remarks

Use this property to enable transparency in the picture specified by the **Picture** property.

You can specify portions of your picture to be transparent, so that an irregularly shaped object does not appear surrounded by a rectangle of solid color. The color that will become transparent is specified by the **PictureMaskColor** property.

PlaySoundFile Method

Applies To

Description

This method causes the control to play the sound file specified.

Syntax

object.**PlaySoundFile** *filename*

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>filename</i>	A string expression that evaluates to the name of a Windows .WAV sound file.

Remarks

The sound file must be distributed along with your application, unless you use a sound that is likely to be on the systems of all your users, such as TADA.WAV.

Note The computer running the application must have a properly installed and configured audio playback device in order to use sounds.

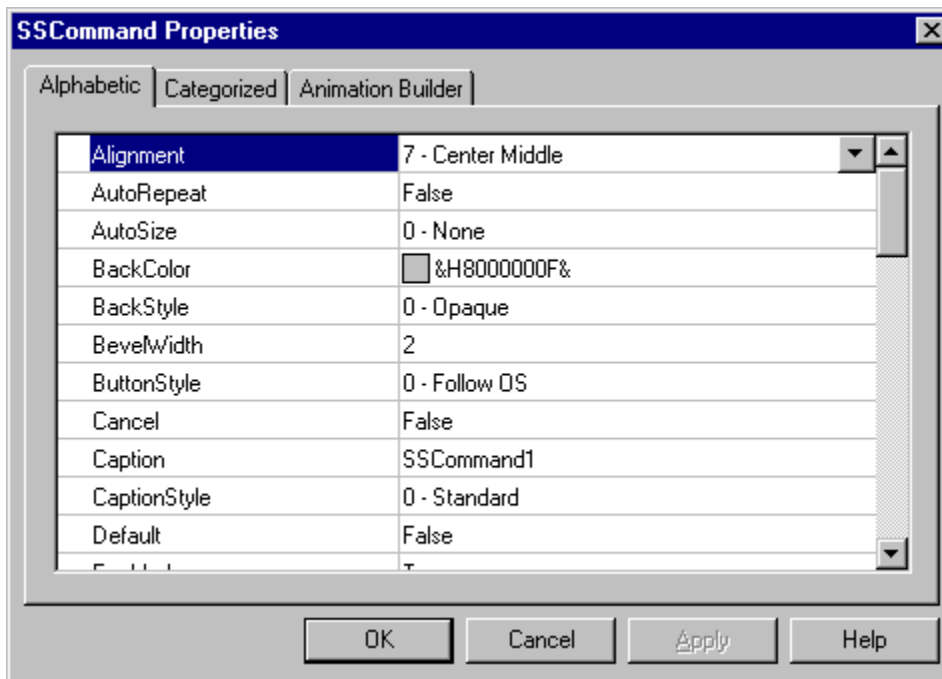
Property Pages

[See Also](#)

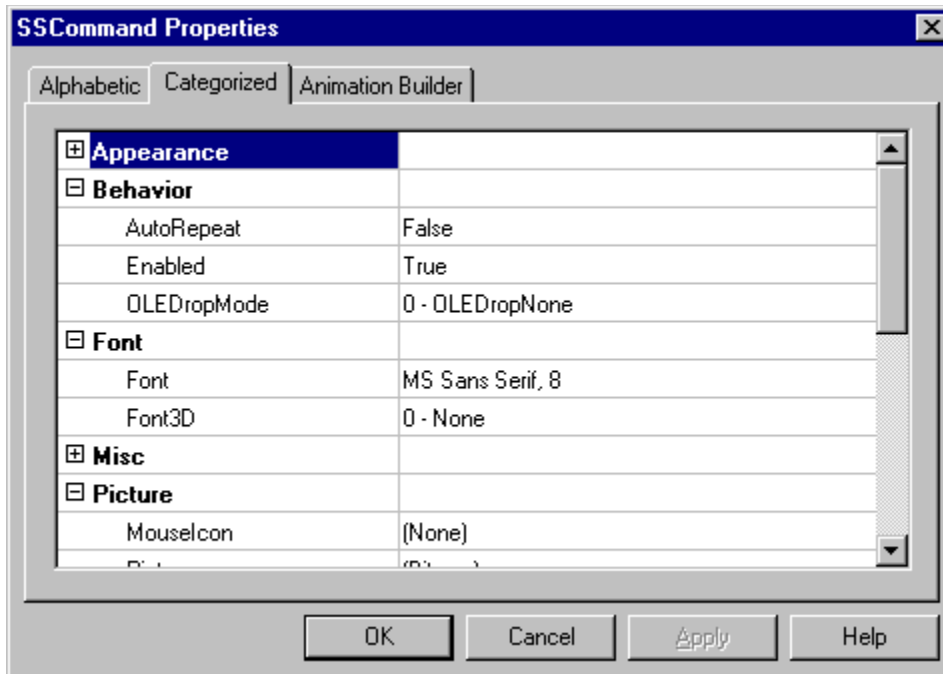
Sheridan Software custom controls support a feature known as property pages. Property pages provide an interface through which you can view and modify the properties of your custom control objects. The purpose of property pages is twofold. First, property pages allow you to set properties at design time that would not otherwise be available - the so-called "runtime" properties. Second, property pages allow you to modify your control in a host environment that does not provide a property sheet.

The Property Pages Interface

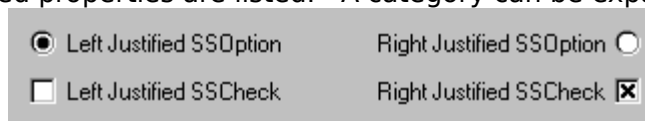
Here is what a Sheridan Property Page looks like:



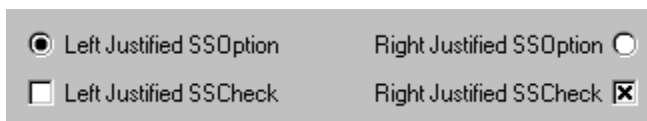
There will always be at least two tabs called 'Alphabetic' and 'Categorized'. The Alphabetic tab contains an alphabetic listing of all properties supported by the control. The Categorized tab contains a list of properties grouped into categories. Take the following property page for example:



In this example the Categorized tab contains categories such as Appearance, Behavior, and Font, under which related properties are listed. A category can be expanded or



collapsed by clicking on the or



to the left of the category name.


Each custom control may also have additional property page tabs. These tabs may contain added functions or utilities. In the picture above, the control also supports a Animation Builder property page which allows the user to perform special functions with the control.

Animated pictures can be created using the [Animation Builder Property Page](#) for ActiveThread's SSFrame, SSPanel, SSOption, SSCheck, SSRibbon and SSCommand controls.

Accessing Property Pages

The method you use to access the property pages of your control depends on two things; the version of the control you are using, and the host environment in which you are using the control.

Many host environments support the use of the secondary mouse button to pop up a context-specific menu. In these environments, you simply click on your control with the secondary mouse button, and choose 'Property Pages' or 'Properties' from the context menu.

If this behavior is not supported, use the property sheet of your design environment. You will see a property labeled '(Custom)' in the property sheet. By double-clicking this property or pressing the  button, you can invoke the property pages for the selected control.

If neither of these methods are supported, you will need to consult the documentation of

your host environment for information on how to change the properties of objects. You may need to choose a special menu option, or perform a shifted mouse-click or double-click on the control. Try searching your environment's online help file for references to objects, embedded objects, object properties, object settings, OLE linking, OLE servers, or properties.

See Also

[Animation Builder Property Page](#)

Quick Tours



Quick Tour 1: Learning To Use ActiveThread Controls

A step-by-step guide to exploring all the features of the SSCheck, SSCommand, SSFrame, SSOption, SSPanel and SSRibbon controls..



Quick Tour 2: Learning To Use The Splitter Control

A step-by-step guide to adding resizable panes to your application using the SSSplitter control.

Refresh Method

Applies To

Description

This method forces a complete repaint of a control.

Syntax

object.**Refresh**

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list..

Remarks

Generally, painting a control is handled automatically while no events are occurring. However, there may be situations where you want the form or control updated immediately, for example, after some external event has caused a change to the form. In such a case, you would use the **Refresh** method.

Remove Method

Applies To

Description

This method removes a pane from the Splitter control.

Syntax

object.**Remove** *panetodelete*

Part	Description
-------------	--------------------

<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list..
---------------	--

<i>panetodelete</i>	A string expression that evaluates to the name of a Pane object in the current Panes collection.
---------------------	--

Remarks

Note You can only remove a pane that does not contain a control. If you attempt to remove a pane containing a control, an error will occur.

Resize Event

[See Also](#)

[Example](#)

[Applies To](#)

Description

Occurs after a splitter bar has moved and all adjacent panes have been resized.

Syntax

Sub *control* **Resize**

Remarks

You can use the **Resize** event to implement specific behaviors that take place whenever the layout of the Splitter control is altered, such as resizing individual panes, or re-positioning controls inside of a container located on a pane.

Resizing Panes Through Code

[See Also](#)

There may be instances when you wish to set the size of a number of panes programmatically. The obvious way to do this is to cycle through the **Panes** collection and set the height and or width of each Pane object. However, using this method may have unpredictable results. The reason for this is that the panes in a Splitter control are organized according to the order in which they were created, and this controls how each individual pane is resized. As you loop through the **Panes** collection, you cannot be sure that resizing one pane will not change the size of other panes that were altered earlier in the loop.

For example, if you have a loop that resizes each pane in a Splitter containing four panes, the loop will resize the panes in order, starting with Panes(0) and ending with Panes(3). Depending on how the panes were created, it is possible that changing the size of Panes(3) will also change the size of Panes(2) or Panes(1). Since the loop has already set these panes to the size you want, any further change to their dimensions is undesirable.

The solution to this problem is to lock each pane in the loop after you have set it to the correct size. Locking the pane prevents the resizing of adjacent panes from altering its dimensions. Once you are finished resizing all the panes, you loop through the collection again and unlock all the panes.

To illustrate this procedure, suppose you have a Splitter control with sixteen panes in a 4 X 4 grid and you want each pane to be the same size. Furthermore, certain panes have their height and/or width locked, and you want them to retain the locked status you set for them at design time. The following code will resize each pane while maintaining its locked status.

First, add this code to the **(General) (declarations)** section of the form containing the Splitter:

```
Private Type PaneLock
    Name As String
    LockHeight As Boolean
    LockWidth As Boolean
End Type
```

Next, add this code to the **Form Load** event

```
Private Sub Form_Load()

    Dim tLockSave() As PaneLock
    Dim LSBWidth As Long
    Dim LPWidth As Long
    Dim LPHeight As Long
    Dim iC As Integer

    ' Determine optimum width of panes
    LSBWidth = SSSplitter1.SplitterBarWidth * Screen.TwipsPerPixelX
    LPWidth = SSSplitter1.Width - (LSBWidth * 3)
    LPWidth = LPWidth / 4

    ' Determine optimum height of panes
    LSBWidth = SSSplitter1.SplitterBarWidth * Screen.TwipsPerPixelY
    LPHeight = SSSplitter1.Height - (LSBWidth * 3)
    LPHeight = LPHeight / 4
```

```

ReDim tLockSave(0 To (SSSplitter1.Panes.Count - 1))
iC = 0

For Each p In SSSplitter1.Panes
    ' Save lock status of pane
    tLockSave(iC).Name = p.Name
    tLockSave(iC).LockHeight = p.LockHeight
    tLockSave(iC).LockWidth = p.LockWidth
    ' Unlock Pane
    p.LockWidth = False
    p.LockHeight = False

    iC = iC + 1
Next p

' Cycle through all panes & set size
For Each p In SSSplitter1.Panes
    ' Set pane to optimum size
    p.Width = lPWidth
    p.Height = lPHeight
    ' Lock pane
    p.LockWidth = True
    p.LockHeight = True
Next p

' Restore original lock status
For iC = 0 To (SSSplitter1.Panes.Count - 1)
    SSSplitter1.Panes(tLockSave(iC).Name).LockWidth =
tLockSave(iC).LockWidth
    SSSplitter1.Panes(tLockSave(iC).Name).LockHeight =
tLockSave(iC).LockHeight
Next iC

End Sub

```

This code uses a dynamically-resized array of user-defined types to keep track of how panes have been locked at design time. The basic structure uses three loops; the first loop saves the lock status of each pane in the array and unlocks the pane. The second loop resizes the panes equally based on the size of the control and the width of the splitter bar. The final loop restores the original lock status of each pane from the array.

RoundedCorners Property

[See Also](#)

[Applies To](#)

Description

Returns or sets a value that determines whether the control will have rounded corners.

Syntax

object.**RoundedCorners** [= *boolean*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying whether the control will have rounded corners, as described in Settings.

Settings

Setting	Description
True	(Default) The corners of the control will appear rounded.
False	The control's corners will be square.

Remarks

This property is ignored when the **ButtonStyle** property is set to anything other than '2 - Windows 3.X Style'

Left Justified SSOption Right Justified SSOption
 Left Justified SSCheck Right Justified SSCheck

SSCheck control

[Properties](#) [Events](#) [Methods](#)

Description

The SSCheck control is a flexible, multimedia-enabled replacement for the standard Windows check box control. An SSCheck control displays a check mark when selected; the check mark disappears when the SSCheck control is cleared. The control may also appear "grayed out" to indicate an indeterminate state.

File Name

THREED20.OCX

Object Type

SSCheck

Remarks

Use this control to give the user a True/False or Yes/No option. Use of the grayed state provides the option of Yes/No/Unsure or True/False/Neither. You can control whether the SSCheck supports two or three states.

You can set the value of an SSCheck control programmatically with the **Value** property. The **Value** property also determines the state of the control, checked; unchecked or grayed.

You can use SSCheck controls in groups to display multiple choices from which the user can select one or more. SSCheck and SSOption controls function similarly but with an important difference: Any number of SSCheck controls on a form can be selected at the same time. In contrast, only one SSOption in a group can be selected at any given time.

In environments that support data binding, the SSCheck can also display the value of a true/false data field from a database.

[← Back](#)

SSCheck Events:

[Click](#)

[KeyDown](#)

[KeyPress](#)

[KeyUp](#)

[MarqueeCycleBegin](#)

[MarqueeCycleEnd](#)

[MouseDown](#)

[MouseEnter](#)

[MouseExit](#)

[MouseMove](#)

[MouseUp](#)

[OLECompleteDrag](#)

[OLEDragDrop](#)

[OLEDragOver](#)

[OLEGiveFeedback](#)

[OLESetData](#)

[OLEStartDrag](#)

Left Justified SSOption

Right Justified SSOption

Left Justified SSCheck

Right Justified SSCheck

SSCheck Methods:

OLEDrag

PlaySoundFile

Refresh

SetFocus

Left Justified SSOption

Right Justified SSOption

Left Justified SSCheck

Right Justified SSCheck

SSCheck Properties:

Entries marked with a * are extender properties.

[\(About\)](#)

[\(Custom\)](#)

[Alignment](#)

[BackColor](#)

[BackStyle](#)

[Caption](#)

[CaptionStyle](#)

[CheckBoxGraphics](#)

[CheckBoxMaskColor](#)

[CheckBoxUseMask](#)

[DataField](#) *

[DataSource](#) *

[DragMode](#)

[Enabled](#)

[Font](#)

[Font3D](#)

[ForeColor](#)

[Height](#)

[HelpContextID](#) *

[Hwnd](#)

[Left](#) *

[MarqueeDelay](#)

[MarqueeDirection](#)

[MarqueeScrollAmount](#)

[MarqueeStyle](#)

[MouseIcon](#)

[MousePointer](#)

[Name](#) *

[OLEDropMode](#)

[Picture](#)

[PictureAlignment](#)

[PictureAnimationDelay](#)

[PictureAnimationEnabled](#)

[PictureFrames](#)

[PictureMaskColor](#)

[PictureUseMask](#)

[TagVariant](#)

[Top](#) *

[TripleState](#)

[Value](#)

Width *
Windowless

Left Justified SSOption Right Justified SSOption
 Left Justified SSCheck Right Justified SSCheck

SSCommand control

[Properties](#) [Events](#) [Methods](#)

Description

The SSCommand control is an extended version of the standard Windows command button. It is functionally identical to a regular command button, but it provides a greater range of interface features, including sound, pictures and animation. You can use an SSCommand control to begin, interrupt, or end a process, or to open or close a dialog box.

File Name

THREED20.OCX

Object Type

SSCommand

Remarks

A user can always choose an SSCommand control by clicking it with the mouse (provided it is not disabled.) Additionally, an SSCommand button can be "pushed" using the spacebar while the control has input focus. To allow the user to choose an SSCommand by pressing ENTER, set its **Default** property to True. To allow the user to choose an SSCommand button by pressing ESC, set the **Cancel** property to True. There can only be one Default and one Cancel button on a form.

Left Justified SSOption

Right Justified SSOption

Left Justified SSCheck

Right Justified SSCheck

SSCommand Events:

Click

KeyDown

KeyPress

KeyUp

MarqueeCycleBegin

MarqueeCycleEnd

MouseDown

MouseEnter

MouseExit

MouseMove

MouseUp

OLECompleteDrag

OLEDragDrop

OLEDragOver

OLEGiveFeedback

OLESetData

OLEStartDrag

Left Justified SSOption

Right Justified SSOption

Left Justified SSCheck

Right Justified SSCheck

SSCommand Methods:

DoClick

OLEDrag

PlaySoundFile

Refresh

SetFocus

Left Justified SSOption

Right Justified SSOption

Left Justified SSCheck

Right Justified SSCheck

SSCommand Properties:

Entries marked with a * are extender properties.

(About)

(Custom)

Alignment

AutoRepeat

AutoSize

BackColor

BackStyle

BevelWidth

ButtonStyle

Cancel *

Caption

CaptionStyle

Default *

DragMode

Enabled

Font

Font3D

ForeColor

Height *

HelpContextID *

Hwnd

Left *

MarqueeDelay

MarqueeDirection

MarqueeScrollAmount

MarqueeStyle

MouseIcon

MousePointer

Name *

OLEDropMode

Outline

Picture

PictureAlignment

PictureAnimationDelay

PictureAnimationEnabled

PictureDisabled

PictureDisabledFrames

PictureDn

PictureDnFrames

PictureFrames

PictureMaskColor

PictureUseMask

RoundedCorners

TagVariant

Top *

Value

Width *

Windowless

Left Justified SSOption Right Justified SSOption
 Left Justified SSCheck Right Justified SSCheck

SSFrame control

[Properties](#)

[Events](#)

[Methods](#)

Description

An SSFrame control is a container control that serves to group other controls both visibly and functionally. For example, you can use an SSFrame control to separate groups of SSOption controls on a single form. Within each SSFrame, the SSOptions operate exclusively of controls outside the SSFrame.

File Name

THREED20.OCX

Object Type

SSFrame

Remarks

To group controls, first draw the SSFrame control, and then draw the controls inside the SSFrame. This enables you to move the SSFrame and the controls it contains together. If you draw a control outside the Frame and then try to move it inside with the mouse, the control will only be on top of the SSFrame. To move a control or controls inside of an SSFrame, you must highlight the control(s), cut them to the clipboard, highlight the SSFrame, then paste the controls from the clipboard.

Left Justified SSOption

Right Justified SSOption

Left Justified SSCheck

Right Justified SSCheck

SSFrame Events:

Click

DbClick

MarqueeCycleBegin

MarqueeCycleEnd

MouseDown

MouseEnter

MouseExit

MouseMove

MouseUp

OLECompleteDrag

OLEDragDrop

OLEDragOver

OLEGiveFeedback

OLESetData

OLEStartDrag

Left Justified SSOption

Right Justified SSOption

Left Justified SSCheck

Right Justified SSCheck

SSFrame Methods:

OLEDrag

PlaySoundFile

Refresh

Left Justified SSOption

Right Justified SSOption

Left Justified SSCheck

Right Justified SSCheck

SSFrame Properties:

Entries marked with a * are extender properties.

(About)

(Custom)

Alignment

BackColor

BackStyle

Caption

CaptionStyle

ClipControls

DragMode

Enabled

Font

Font3D

ForeColor

Height *

HelpContextID *

Hwnd

Left *

MarqueeDelay

MarqueeDirection

MarqueeScrollAmount

MarqueeStyle

MouseIcon

MousePointer

Name *

OLEDropMode

Picture

PictureAlignment

PictureAnimationDelay

PictureAnimationEnabled

PictureBackground

PictureBackgroundStyle

PictureFrames

PictureMaskColor

PictureUseMask

ShadowStyle

TagVariant

Top *

Width *



SSOption control

[Properties](#)

[Events](#)

[Methods](#)

Description

The SSOption control is a flexible, multimedia-enabled replacement for the standard Windows option button control. SSOption controls operate in exclusive mode - when one SSOption is selected, any others in the same container are automatically deselected.

File Name

THREED20.OCX

Object Type

SSOption

Remarks

Usually, SSOption controls are used in a group to display options from which the user can select only one. You group SSOption controls by drawing them inside a container such as an SSFrame control, an SSPanel control, or a form.

While SSOption controls and SSCheck controls may appear to function similarly, there is an important difference: When a user selects an SSOption, the other SSOption controls in the same group are automatically unselected. In contrast, any number of SSCheck controls can be selected at the same time.

Left Justified SSOption

Right Justified SSOption

Left Justified SSCheck

Right Justified SSCheck

SSOption Events:

Click

DbClick

KeyDown

KeyPress

KeyUp

MarqueeCycleBegin

MarqueeCycleEnd

MouseDown

MouseEnter

MouseExit

MouseMove

MouseUp

OLECompleteDrag

OLEDragDrop

OLEDragOver

OLEGiveFeedback

OLESetData

OLEStartDrag

Left Justified SSOption

Right Justified SSOption

Left Justified SSCheck

Right Justified SSCheck

SSOption Methods:

OLEDrag

PlaySoundFile

Refresh

SetFocus

Left Justified SSOption

Right Justified SSOption

Left Justified SSCheck

Right Justified SSCheck

SSOption Properties:

Entries marked with a * are extender properties.

[\(About\)](#)

[\(Custom\)](#)

[Alignment](#)

[BackColor](#)

[BackStyle](#)

[Caption](#)

[CaptionStyle](#)

[DragMode](#)

[Enabled](#)

[Font](#)

[Font3D](#)

[ForeColor](#)

[Height](#) *

[HelpContextID](#) *

[Hwnd](#)

[Left](#) *

[MarqueeDelay](#)

[MarqueeDirection](#)

[MarqueeScrollAmount](#)

[MarqueeStyle](#)

[MouseIcon](#)

[MousePointer](#)

[Name](#) *

[OLEDropMode](#)

[OptionBtnGraphics](#)

[OptionBtnMaskColor](#)

[OptionBtnUseMask](#)

[Picture](#)

[PictureAlignment](#)

[PictureAnimationDelay](#)

[PictureAnimationEnabled](#)

[PictureFrames](#)

[PictureMaskColor](#)

[PictureUseMask](#)

[TagVariant](#)

[Top](#) *

[Value](#)

[Width](#) *

[Windowless](#)

Left Justified SSOption Right Justified SSOption
 Left Justified SSCheck Right Justified SSCheck

SSPanel control

[Properties](#)

[Events](#)

[Methods](#)

Description

An SSPanel control is a container control that serves to group other controls both visibly and functionally. The SSPanel provides a unified, distinctive area suited to the creation of control groupings such as toolbars and control panels. The SSPanel can also function as a progress indicator, showing the percentage of completion of a task both graphically and textually.

File Name

THREED20.OCX

Object Type

SSPanel

Remarks

The SSPanel is a three-dimensional rectangular area of variable size that can be as large as the form itself or just large enough to display a single line of text. It can present status information in a dynamically colored circle or bar with or without showing percent.

You can create a number of interesting 3-D border effects with the panel by changing the type of beveled borders used on the outside of the control.

In environments that support data binding, the SSPanel can also display text or numeric data from a database.

Left Justified SSOption

Right Justified SSOption

Left Justified SSCheck

Right Justified SSCheck

SSPanel Events:

Click

DbClick

MarqueeCycleBegin

MarqueeCycleEnd

MouseDown

MouseEnter

MouseExit

MouseMove

MouseUp

OLECompleteDrag

OLEDragDrop

OLEDragOver

OLEGiveFeedback

OLESetData

OLEStartDrag

Left Justified SSOption

Right Justified SSOption

Left Justified SSCheck

Right Justified SSCheck

SSPanel Methods:

OLEDrag

PlaySoundFile

Refresh

Left Justified SSOption

Right Justified SSOption

Left Justified SSCheck

Right Justified SSCheck

SSPanel Properties:

Entries marked with a * are extender properties.

(About)

(Custom)

Align

Alignment

AutoSize

BackColor

BackStyle

BevelInner

BevelOuter

BevelWidth

BorderWidth

Caption

CaptionStyle

ClipControls

DataField *

DataSource *

DragMode

Enabled

FloodColor

FloodFillStyle

FloodPercent

FloodShowPct

FloodType

Font

Font3D

ForeColor

Height *

HelpContextID *

Hwnd

Left *

MarqueeDelay

MarqueeDirection

MarqueeScrollAmount

MarqueeStyle

Mouselcon

MousePointer

Name *

OLEDropMode

Outline

Picture

PictureAlignment

PictureAnimationDelay

PictureAnimationEnabled

PictureBackground

PictureBackgroundStyle

PictureFrames

PictureMaskColor

PictureUseMask

RoundedCorners

TagVariant

Top *

Width *



SSRibbon control

[Properties](#)

[Events](#)

[Methods](#)

Description

The SSRibbon control is similar to the SSCommand button, except that toggles between two states (up and down) when clicked. It also displays the same exclusive behavior as an option button, but its exclusivity is determined by a property of the control rather than its grouping inside a container.

File Name

THREED20.OCX

Object Type

SSRibbon

Remarks

SSRibbon buttons can be used in groups to emulate the functionality of a toolbar or ribbon such as those found in Microsoft Excel and Microsoft Word. Unlike the SSCommand button, the SSRibbon button cannot receive input focus; it must be clicked with the mouse.

Left Justified SSOption

Right Justified SSOption

Left Justified SSCheck

Right Justified SSCheck

SSRibbon Events:

Click

MarqueeCycleBegin

MarqueeCycleEnd

MouseDown

MouseEnter

MouseExit

MouseMove

MouseUp

OLECompleteDrag

OLEDragDrop

OLEDragOver

OLEGiveFeedback

OLESetData

OLEStartDrag

Left Justified SSOption

Right Justified SSOption

Left Justified SSCheck

Right Justified SSCheck

SSRibbon Methods:

OLEDrag

PlaySoundFile

Refresh

Left Justified SSOption

Right Justified SSOption

Left Justified SSCheck

Right Justified SSCheck

SSRibbon Properties:

Entries marked with a * are extender properties.

(About)

(Custom)

Alignment

AutoSize

BackColor

BackStyle

BevelWidth

ButtonStyle

Caption

CaptionStyle

DragMode

Enabled

Font

Font3D

ForeColor

GroupAllowAllUp

GroupNumber

Height *

HelpContextID *

Hwnd

Left *

MarqueeDelay

MarqueeDirection

MarqueeScrollAmount

MarqueeStyle

Mouselcon

MousePointer

Name *

OLEDropMode

Outline

Picture

PictureAlignment

PictureAnimationDelay

PictureAnimationEnabled

PictureDisabled

PictureDisabledFrames

PictureDn

PictureDnChange

PictureDnDisabled

PictureDnDisabledFrames

PictureDnFrames

PictureFrames

PictureMaskColor

PictureUseMask

RoundedCorners

TagVariant

Top *

Value

Width *

Windowless

Left Justified SSOption Right Justified SSOption
 Left Justified SSCheck Right Justified SSCheck

SSSplitter control

[Properties](#)

[Events](#)

[Methods](#)

[Objects](#)

[Collections](#)

Description

The SSSplitter is a container control that organizes the controls it contains into resizable panes, which are separated by splitter bars that can be dragged with the mouse. It provides functionality similar to that of the frames found on web pages that use the HTML 2.0 (or greater) specification.

File Name

SPLITTER.OCX

Object Type

SSSplitter

Remarks

The SSSplitter control consists of a collection of **Pane** objects. Each pane object can hold one control, and that control is automatically resized to fill the area of the pane whenever the pane itself is resized. Objects inside the SSSplitter behave as if they were inside a single container; for example, SSOption buttons on separate panes still operate exclusively with SSOptions on other panes.

Left Justified SSOption

Right Justified SSOption

Left Justified SSCheck

Right Justified SSCheck

SSSplitter Events:

Click

DbClick

MouseDown

MouseMove

MouseUp

Resize

SplitterEndDrag

SplitterStartDrag

Left Justified SSOption

Right Justified SSOption

Left Justified SSCheck

Right Justified SSCheck

SSSplitter Methods:

[PaneFromControl](#)

[PaneFromPosition](#)

[PaneFromPositionEx](#)

[PlaySoundFile](#)

[Refresh](#)

[Remove](#)

Left Justified SSOption

Right Justified SSOption

Left Justified SSCheck

Right Justified SSCheck

SSSplitter Properties:

Entries marked with a * are extender properties.

(About)

(Custom)

Align

AutoSize

BackColor

BorderStyle

ClipControls

DragMode

Enabled

Height *

Hwnd

Left *

Locked

Name *

SplitterBarAppearance

SplitterBarWidth

SplitterBarJoin

SplitterResizeStyle

TagVariant

Top *

Width *

Special Thanks To

Felicia, Bryan, Michele and Jessica

Special Thanks To

Monica

Special Thanks To

Kym

Special Thanks To

Eileen & Clyde

Special Thanks To

Lori

Special Thanks To

Lisa Joy

Special Thanks To

Alim, Timor, Dilshod, The Professor & The Weasel

Special Thanks To

Kim and T.J.

Special Thanks To

Barbara, Daniel, Lauren and baby James

SetData Method

[See Also](#)

[Applies To](#)

Description

This method is used to insert data into an **ssDataObject** object using the specified data format.

Syntax

object.**SetData** [*data*], [*format*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>data</i>	Optional. A variant containing the data to be passed to the ssDataObject object.
<i>format</i>	Optional. An integer expression specifying the format of the data being passed, as described in Settings.

Settings

The settings for *format* are:

Setting	Description
1	Text (.TXT files)
2	Bitmap (.BMP files)
3	metafile (.WMF files)
8	Device-independent bitmap (DIB)
9	Color palette
14	Enhanced metafile (.EMF files)
15	List of files
-16639	Rich text format (.RTF files)

Remarks

Constants that correspond to these formats are available. See [Constants](#) for more information.

The *data* argument is optional. This allows you to set several different formats that the source component can support without having to load the data separately for each format. Multiple formats are set by calling **SetData** several times, each time using a different format. If you wish to start fresh, use the Clear method to clear all data and format information from the **ssDataObject**.

The *format* argument is also optional, but either the data or format argument must be specified. If data is specified, but not format, then your development environment should try to determine the format of the data. If it is unsuccessful, then an error is generated. When the target requests the data, and a format was specified, but no data was provided, the source's **OLESetData** event occurs, and the source can then provide the requested data type.

It's possible for the **GetData** and **SetData** methods to use data formats other than those listed in Settings, including user-defined formats registered with Windows via the RegisterClipboardFormat() API function. However, there are a few caveats:

- § The **SetData** method requires the data to be in the form of a byte array when it does not recognize the data format specified.
- § The **GetData** method always returns data in a byte array when it is in a format that it doesn't recognize, although Visual Basic can transparently convert this returned byte array into other data types, such as strings.
- § The byte array returned by **GetData** will be larger than the actual data when running on some operating systems, with arbitrary bytes at the end of the array. The reason for this is that the application does not always know the data's format, and knows only the amount of memory that the operating system has allocated for the data. This allocation of memory is often larger than is actually required for the data. Therefore, there may be extraneous bytes near the end of the allocated memory segment. As a result, you must use appropriate functions to interpret the returned data in a meaningful way (such as truncating a string at a particular length with the Left function if the data is in a text format).

Note Not all applications support `ssCFBitmap` or `ssCFPalette`, so it is recommended that you use `ssCFDIB` whenever possible.

SetFocus Method

Applies To

Description

This method moves the focus to the specified control..

Syntax

object.**SetFocus**

Part	Description
-------------	--------------------

<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list..
---------------	--

Remarks

This is an [extender](#) method.

The object must be a control that can receive the focus. After invoking the **SetFocus** method, any user input is directed to the specified control.

You can only move the focus to a visible control. You also can't move the focus to a control if the **Enabled** property is set to False. If the **Enabled** property has been set to False at design time, you must first set it to True before it can receive the focus using the **SetFocus** method.

ShadowStyle Property

[See Also](#)

[Applies To](#)

Description

Returns or sets a value that specifies how the frame will be drawn.

Syntax

object.**ShadowStyle** [= *number*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the frame style, as described in Settings.

Settings

Setting	Description
0	(Default) Inset. The frame appears sunken into the background.
1	Raised. The frame appears raised above the background.

Remarks

This property determines the three-dimensional appearance of the border of the Frame control. If inset, the frame is highlighted and shadowed so as to appear sculpted in to the background. If raised, the frame appears as a ridge that rises above the background.

Size Property

[See Also](#)

[Applies To](#)

Description

Returns or sets the font size used in the specified **Font** object.

Syntax

object.**Size** [= *number*]

The **Size** property syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the size of the font in points.

Remarks

Use this property to format text in the font size you want. The default font size is determined by the operating system. To change the default, specify the size of the font in points. The maximum value for the **Size** property is 2048 points.

The **Font** object is not directly available at design time. Instead you set the **Size** property through a control's **Font** property.

At run time, however, you can set **Size** directly by specifying its setting for the appropriate **Font** object.



Felicia, Bryan, Michele & Jessica

Eileen & Clyde

**Alim, Timor, Dilshod, The Professor
& The Weasel**

Lisa Joy

Barbara, Daniel, Lauren & baby James

**Lori
Monica
Kim & T.J.
Kym**

**Special thanks for
their support!**

Left Justified SSOption

Right Justified SSOption

Left Justified SSCheck

Right Justified SSCheck

SplitterBarAppearance Property

[See Also](#)

[Applies To](#)

Description

Returns or sets a value that specifies the 3-D look of the splitter bars.

Syntax

object.**SplitterBarAppearance** [= *number*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the appearance of the splitter bars, as described in Settings.

Settings

Setting	Description
0	Borderless
1	Flat
2	(Default) 3-D (beveled)

SplitterBarJoinStyle Property

[See Also](#)

[Applies To](#)

Description

Returns or sets a value that specifies how the splitter bars connect to one another.

Syntax

object.**SplitterBarJoinStyle** [= *number*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the appearance of the splitter bar junctions, as described in Settings.

Settings

Setting	Description
0	(Default) Segmented. A visible seam is displayed at the junction between horizontal and vertical splitter bars.
1	Continuous. Splitter bars blend into one another at the junctions.

SplitterBarWidth Property

[See Also](#)

[Applies To](#)

Description

Returns or sets the width of the splitter bars in pixels

Syntax

object.**SplitterBarWidth** [= *number*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying the width of the splitter bars.

Remarks

If **SplitterBarWidth** is set to a value too small to display a 3-D or bordered effect (less than 3 pixels,) the setting of **SplitterBarAppearance** will be ignored.

You can set the **SplitterBarWidth** to 0. This will create invisible splitter bars that cannot be moved by the user.

SplitterEndDrag Event

[See Also](#)

[Example](#)

[Applies To](#)

Description

Occurs after a splitter bar has been moved, but before the panes are resized in response to the move.

Syntax

Sub *control*_**SplitterEndDrag**

Remarks

You can use this event to determine whether and how individual panes will be resized in response to a splitter bar drag. You can also include code to place limitations on splitter bar movement.

SplitterResizeStyle Property

[See Also](#)

[Applies To](#)

Description

Returns or sets a value that specifies how the individual panes will behave when the entire Splitter control is resized.

Syntax

object.**SplitterResizeStyle** [= *number*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An integer expression specifying how the Splitter will respond to resizing, as described in Settings.

Settings

Setting	Description
0	Splitter panes will not resize when the control is sized. The panes along the moving edge of the control will be expanded or clipped to accommodate the new control size.
1	Splitter panes will resize proportionally when the control is sized. All panes will grow or shrink, either horizontally or vertically, to accommodate the new control size.

Remarks

This property is particularly useful when the **AutoSize** property is set to Fill Container and the Splitter control is on a form. When the form is resized, all the splitter panes will automatically be proportionally resized as well.

SplitterStartDrag Event

[See Also](#)

[Example](#)

[Applies To](#)

Description

Occurs when a splitter bar begins to move in response to the user dragging it.

Syntax

Sub *control_SplitterStartDrag* ([*index As Integer*] *SplitterBarType As Long*, *BorderPanels As Panes*, *Cancel As Boolean*)

The event parameters are:

Parameter Description

<i>index</i>	An integer expression that uniquely identifies a control if it is in a control array.
<i>SplitterBarType</i>	An integer expression that evaluates to the direction of the movement, as described in Settings.
<i>BorderPanels</i>	A collection object consisting of the Pane objects adjacent to the moving splitter bar, and thus affected by it. This is a collection of all the panes that will be resized in response to the move.
<i>Cancel</i>	A boolean expression that determines if the drag operation will be completed. Setting <i>Cancel</i> to True stops the drag operation, and no change is made to the control.

Settings

Setting Description

0	Horizontal. The splitter bar being moved is a horizontal splitter bar.
1	Vertical. The splitter bar being moved is a vertical splitter bar.
2	Both. A junction between two splitter bars is being moved

Remarks

You can use this event to provide a fine degree of control of the resizing of panes in your Splitter control. You can set minimum, maximum and fixed horizontal and vertical sizes on a pane-by-pane basis by examining the panes which will be affected by the resize operation before it occurs.

Strikethrough Property

[See Also](#)

[Applies To](#)

Description

Returns or sets the font style of the specified **Font** object to either strikethrough or non-strikethrough.

Syntax

object.**Strikethrough** [= *boolean*]

The **Strikethrough** property syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A Boolean expression specifying the font style as described in Settings.

Settings

The settings for *boolean* are:

Setting	Description
True	Turns on strikethrough formatting.
False	(Default) Turns off strikethrough formatting.

Remarks

The **Font** object is not directly available at design time. Instead you set the **Strikethrough** property through a control's **Font** property.

At run time, however, you can set **Strikethrough** directly by specifying its setting for the appropriate **Font** object.

System Requirements

You must have the following to run this product.

- § A hard disk with approximately 15 megabytes of available space for a full installation, including all documentation and sample files. For just the controls and system DLLs, less than 2 megabytes is required.
- § At least four megabytes of RAM. Some environments may require more than four megabytes.
- § Windows 95 or later, or Windows NT 3.51 or later. If you are using Windows NT 3.51, you must have Service Pack 5 or greater installed.
- § A host environment that fully supports the ActiveX control specifications
- § Container controls (SSFrame, SSPanel & SSSplitter) require a host environment that supports the ISimpleFrameSite interface (such as Visual Basic) in order to contain child controls.
- § Windowless controls (SSCheck, SSCommand, SSOption, SSRibbon) require a host environment that supports the IWindowlessControlSite interface (such as Visual Basic 5.0) in order to operate in windowless mode. In all other environments, the controls will operate in windowed mode; i.e. they will have a window handle.

TagVariant Property

Applies To

Description

Returns or sets any extra data needed for your program. You can use this property to attach data of any type, except user defined types, to an object or control.

Syntax

object.**TagVariant** [= *expression*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>expression</i>	A variant expression

Remarks

The **TagVariant** property is similar to the Visual Basic **Tag** property. However, in addition to string expressions, the **TagVariant** property can store any data type including other objects.

Note The TagVariant property can store all data types except user defined types.

Technical Specifications



Property Pages

A guide to using the Property Pages to set features of the control, including the properties of sub-objects and animated pictures.



Troubleshooting & Tips

Procedures to solve the most common problems with the product



Compatibility Issues

Procedures to solve the most common problems with the product



System Requirements

A list of requirements for using the product



Included Files

A list of files included with the product and their locations



Distribution Notes

A list of the files you need to distribute with your applications

Top Property

[See Also](#)

[Applies To](#)

Description

Returns or sets the distance between the top edge of an object and the top edge of its container.

Syntax

`object.Top [= number]`

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An single precision expression specifying the distance of the object from the top edge of its container. The value returned is in the scale units of the object's container.

Remarks

This is an [extender](#) property, except when applied to the **Pane** object..

Use the **Height**, **Width**, **Left**, and **Top** properties for operations or calculations based on an object's total area, such as sizing or moving the object.

This property is read-only for the **Pane** object.

Trappable Errors

Trappable errors can occur while an application is running, either within the development environment or as a stand-alone executable. Some of these can also occur during design time or compile time.

General Errors

Code	Definition
380	Invalid Property Value
30009	The BackStyle can not be set to transparent at run-time if it was not set to Transparent at design-time.

SSPanel Errors

Code	Definition
30002	Bevel width must be from 0 to 30
30003	Border width must be from 0 to 30
30006	Flood percent must be from 0 to 100

SSCommand/SSRibbon Errors

Code	Definition
30004	Bevel width must be from 0 to 10
30005	Group number must be from 0 to 99

SSSplitter Errors

Code	Definition
31001	Invalid position passed to PaneFromPosition method. The point may lie on a splitter bar or outside of the Splitter control's client area.
31002	Specified control is invalid. You passed a value that is not a control name, and a control name was expected.
31003	PaneFromControl could not locate the pane based on the argument control
31004	Tried to remove the root pane in the Panes collection (via the Remove method)
31005	Tried to remove a pane that has a control in it via the remove method
31006	Invalid control name
31007	The Add method of the Panes collection was passed a pane name which contains a control.
31008	Add method of Panes collection passed a bad pane name to split
31009	Pane too small. You tried to add a pane through code, but the resulting pane would have been too small to display.
31010	Add method of Panes collection passed a new pane name that already exists.

TripleState Property

[See Also](#)

[Applies To](#)

Description

Returns or sets a value that specifies whether the control will have two or three logical states.

Syntax

object.**TripleState** [= *boolean*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying number of states of the control, as described in Settings.

Settings

Setting	Description
True	Control will have three logical states: Checked, Unchecked and Grayed.
False	(Default) Control will have two logical states: Checked and Unchecked.

Remarks

This property affects the settings of the **Value** property of the control. When **TripleState** is FALSE any non-zero value means checked and 0 means unchecked. When **TripleState** is TRUE, a **Value** of 0 means Unchecked, 1 means Checked and 2 means Grayed. Any other non-zero value also evaluates to Checked.

You can use the **TripleState** property to ensure compatibility with either earlier versions of the THREED check box control or the standard Windows check box control.

Troubleshooting and Tips

As controls get more complicated and must operate in a wider range of development and operating environments, the complexity of the interaction between operating system, programming system and custom control increases. With all this complexity, things may occasionally go wrong. There might also be useful features of the product that you are not using because you don't know they are there.

Below is a list of helpful suggestions and common problems you may encounter when using the product and writing and distributing applications that use it. Click the description which most closely matches the information you want or the problem you are having.

When I started using the SSSplitter, I saw a QuickHelp screen with tips on how to use the control. Now that screen is gone. How do I get it back?

I used a particular color as a mask color in a graphic, but when I specify that color for the MaskColor property, it appears in the control and is not masked out. Why?

Left Justified SSOption Right Justified SSOption
 Left Justified SSCheck Right Justified SSCheck

I would like to use an SSRibbon

button as a command button - that is, when you click it, it pops right back up. Is there a way to do this?

Left Justified SSOption Right Justified SSOption
 Left Justified SSCheck Right Justified SSCheck

How can I make buttons that look like

those in Internet Explorer - gray until the mouse passes over them, then becoming colorful?

Left Justified SSOption Right Justified SSOption
 Left Justified SSCheck Right Justified SSCheck

I specified a picture for the SSFrame

or SSPanel, but it's too small or in the wrong place. How can I add a big picture or texture to these controls?

Left Justified SSOption Right Justified SSOption
 Left Justified SSCheck Right Justified SSCheck

I have written some code to resize

the panes in my SSSplitter control, but the results do not come out as I expected. How do I resize panes through code?

Left Justified SSOption Right Justified SSOption
 Left Justified SSCheck Right Justified SSCheck

The colors do not appear right in my

controls that have pictures, or the colors on my system change when I use the controls. What is going on?

<input checked="" type="radio"/> Left Justified SSOption	Right Justified SSOption <input type="radio"/>
<input type="checkbox"/> Left Justified SSCheck	Right Justified SSCheck <input checked="" type="checkbox"/>

Troubleshooting Answers

When I started using the SSSplitter, I saw a QuickHelp screen with tips on how to use the control. Now that screen is gone. How do I get it back?

ActiveThread provides QuickHelp when you first begin using the SSSplitter control. QuickHelp is a summary of the control's major features and how to get started using them. After you are familiar with using the Splitter, you will probably want to turn off QuickHelp. You can do this by checking the box at the bottom of the QuickHelp screen that begins "Don't show me this dialog again..."

If you've turned off QuickHelp and want to get it back, simply do the following:

1. Create a new project and add an SSSplitter control to your form.
2. Click the Splitter with your secondary mouse button to bring up the context menu for the control.
3. From the menu select "QuickHelp..." The QuickHelp screen will appear. At this point you can read the information on the screen. QuickHelp is always available via this menu option.
4. If you want QuickHelp to appear every time you create a new Splitter control, clear the check box at the bottom of the screen that says "Don't show me this dialog again..."
5. When you are finished with QuickHelp, click the OK button to close the screen.

I used a particular color as a mask color in a graphic, but when I specify that color for the MaskColor property, it appears in the control and is not masked out. Why?

First you must be sure you have specified a mask color for the control and that color masking is enabled. Check to see that you have correctly set the **PictureMaskColor** (or **OptionBtnMaskColor** or **CheckBoxMaskColor**) property to the correct color, and that the **PictureUseMask** (or **OptionBtnUseMask** or **CheckBoxUseMask**) property is set to True. If this does not solve the problem there may be another cause.

Different types of video drivers seem to handle color values differently in different situations. This can mean that although you use a certain color in your bitmap, and then specify that same color as the mask color in your ActiveThread control, the color will not be masked and will appear in the bitmap. Even if you successfully mask out the color, you may find that the color does not appear masked if you move your application to another machine with a different video system.

To be on the safe side, we recommend that you use one of the primary colors - red, green or blue, at full intensity - for your mask color. In terms of RGB values this would be RGB(255,0,0), RGB(0,255,0) or RGB(0,0,255). When you create the bitmap in your bitmap editor, make sure to use one of these colors for the areas you want to appear as transparent. Of the three, red seems to work best.

One other consideration is the number of colors in your bitmap. If you are using high-color bitmaps (bitmaps saved in a mode that supports 65536 or more colors) and you continue to experience problems with mask colors, try converting your bitmaps to 256 colors. Most image editors can perform this type of conversion. If your application is going to be run on

256-color systems, see the topic below on [using 256-color bitmaps](#).

Take the following steps to correct this situation:

1. Open the bitmap in your bitmap editing program.
2. Using the editor's flood fill tool, fill in all the areas that you want to appear transparent with one of the preferred mask colors, red, blue, or green. Try red first. Make sure that the colors are pure; i.e 100% of the specified color.
3. Save the bitmap using your editor's Save command. You may want to use Save As and give the bitmap a different name.
4. Load the new bitmap into your ActiveThread control by setting one of the control's picture-related properties (**Picture**, **PictureDn**, **PictureDnDisabled**, **PictureBackground**) to the filename of the bitmap file.
5. Set the **PictureMaskColor** property of the control to the color you used in the bitmap as the mask color. Make sure the **PictureUseMask** property is set to True

I would like to use an SSRibbon button as a command button - that is, when you click it, it pops right back up. Is there a way to do this?

The SSRibbon button is not designed to operate in command-button mode. If you need a command button, you should use the SSCommand control. Because the SSRibbon button cannot receive input focus, it will not display a focus rectangle and you cannot use the keyboard to select or click the button. This violates standard Windows interface conventions and may confuse the user of your application.

However, if for some reason you want to use an SSRibbon button as a command button substitute, you can implement code in the control's events to make it respond in a manner similar to a regular command button. Keep in mind, though, that the functionality may not be identical to that of the SSCommand button.

To use an SSRibbon button as a command button, take the following steps:

1. Add the SSRibbon button to a form in your project.
2. Place the following code in the **Click** event of the SSRibbon button. Substitute the name of your control for SSRibbon1:

```
SSRibbon1.Value = 0
```

How can I make buttons that look like those in Internet Explorer - gray until the mouse passes over them, then becoming colorful?

The SSCommand and SSRibbon button contain properties that you can use to specify different pictures for the various states of the control, and you can create Internet Explorer-style borders on your buttons by simply setting the **ButtonStyle** property of the control to '3 - ActiveBorders.' However, if you want to duplicate the color-changing functionality of Internet Explorer's buttons, you need to use code.

The SSCommand and SSRibbon controls feature two events that make it easy to add this capability to your programs: **MouseEnter** and **MouseExit**. By placing code in these events to change the picture-related properties of the control, you can make your buttons even more mouse-sensitive.

To do this you will need two copies of each button picture you use; a color version and a grayscale or monochrome version. You make these pictures available to the application, then switch between them in response to the mouse events.

To change the button picture when the mouse passes over the button, take the following steps:

1. In the **MouseEnter** event of your control, add the following code. The filenames of the pictures are provided as examples only. You could alternately specify the **Picture** property of another control in your application (perhaps an invisible picture box, image control, or an image list control.)

```
' Change the button pictures to their color versions
Set SSRibbon1.Picture = LoadPicture("COLORUP.BMP")
Set SSRibbon1.PictureDn = LoadPicture("COLORDN.BMP")
```

2. Place the following code in the **MouseExit** event of the control:

```
' Change the button pictures to their grayscale versions
Set SSRibbon1.Picture = LoadPicture("GRAYUP.BMP")
Set SSRibbon1.PictureDn = LoadPicture("GRAYDN.BMP")
```

You might want to additionally add code to the **MouseEnter** events of other buttons on the form that would change the pictures of all buttons other than themselves to their grayscale versions. This will often solve problems that occur when the user moves the mouse extremely rapidly across the controls.

I specified a picture for the SSFrame or SSPanel, but it's too small or in the wrong place. How can I add a big picture or texture to these controls?

The container controls of ActiveThread (SSFrame and SSPanel) support a special picture property called **PictureBackground**. For these controls, the standard **Picture** property controls the picture that will be displayed in the caption area of the control. (For the SSPanel, this is effectively the entire area of the control.) If you wish to display a large picture that appears behind the controls in the container, or that will act as "wallpaper" for the control, you must use the **PictureBackground** property. Use of the **PictureBackground** gives you added ability to manipulate the picture, such as stretching or tiling it to fill the control. **PictureBackground** also operates without regard to any caption the control might have or the settings of any of the Alignment properties.

Note that **PictureBackground** uses the same mask color properties as the regular **Picture** property. This makes it possible to have transparent areas in your control's background as well as in its caption picture.

I have written some code to resize the panes in my SSSplitter control, but the results do not come out as I expected. How do I resize panes through code?

Panes are resized based on the order in which they were created. When looping through the Panes collection in code, you may find that resizing panes in collection order does not produce the results you want. This is because resizing of panes later in the loop may inadvertently change the size of panes set up earlier in the loop.

The best way to manage the resizing of panes through code is to lock each pane as it is resized. This way, later changes to adjacent panes will not affect the size of panes you have already set. The procedure for doing this, which includes the ability to retain the design-time **LockHeight** and **LockWidth** settings of individual panes, is outlined in [Resizing Panes Through Code](#).

The colors do not appear right in my controls that have pictures, or the colors on my system change when I use the controls. What is going on?

This is a common problem on systems using a 256-color display adapter. The system can only display 256 colors at a time. Windows attempts to compensate for colors that are outside its displayable range by changing existing colors or switching the entire palette of colors that the system is using. When this "palette shift" occurs, you may see the graphics on your screen flash or change color completely. Systems that use more than 256 colors do not exhibit palette shift.

If you use high color images in your application (also known as 16-bit color and 24-bit color images) users of 256-color systems will observe palette shifts when your application is run on their system. This is an unavoidable side effect of Windows attempting to compensate for the lack of available colors when displaying your graphics.

The best way to minimize the effects of palette shift on 256-color systems is to use 256-color pictures that use a common color palette. This will at least insure that the pictures in your application do not conflict with one another, although you may still observe palette shifting in graphics outside of your application.

Many image editing programs contain tools for modifying and converting color palettes. Your best approach is to standardize on a single palette for your applications, then use this palette to create all the graphics for your program. Any clip art you use, or graphics you have created previously, can be converted to this palette. If necessary, pictures can be dithered to adapt their palettes to the one you are using.

You may also find it helpful to use "identity" palettes. An identity palette is a special palette that has the high intensity VGA colors as the first 8 entries in its palette, and the low-intensity VGA colors as the last 8 entries. Identity palettes can minimize palette shift in colors outside of your application, particularly those used by Windows itself. Again, you will need an image editor or conversion tool that supports the creation of identity palettes.

Another possibility is to use images that contain less than 256 colors. Often, you will find that dithering an image to 128 colors or even 64 colors provides acceptable quality. If you use an identity palette with a sub-256-color image, you may find you are able to eliminate palette shift altogether. In addition, the size of your graphics and executable files will be smaller.

One final solution is to use your image editor to convert your graphics all the way down to the 16-color VGA palette. This may result in pictures that are grainy or unclear, but with certain images and careful dithering, the results can look surprisingly good. This approach will eliminate any palette shift considerations as well as making your application suitable for use on 16 color VGA systems.

Underline Property

[See Also](#)

[Applies To](#)

Description

Returns or sets the font style of the specified **Font** object to either underlined or non-underlined.

Syntax

object.**Underline** [= *boolean*]

The **Underline** property syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A Boolean expression specifying the font style as described in Settings.

Settings

The settings for *boolean* are:

Setting	Description
True	Turns on underlining.
False	Turns off underlining.

Remarks

The **Font** object is not directly available at design time. Instead you set the **Underline** property through a control's **Font** property.

At run time, however, you can set **Underline** directly by specifying its setting for the appropriate **Font** object.

Upgrade Notes

This section contains important information for users upgrading from earlier versions of THREED to ActiveThread.

Note ActiveThread is a 32-bit only product! It will not work with the 16-bit version of Visual Basic 4.0

When upgrading a Visual Basic project that uses THREED.VBX the conversion process is automatic. After you have installed THREED20.OCX, any projects containing THREED.VBX will be automatically upgraded to the newer version when they are opened in the Visual Basic development environment.

For Visual Basic 4.0 32-bit

It is possible to upgrade a Visual Basic 4.0 project directly from THREED16.OCX or THREED32.OCX to THREED20.OCX by editing the project (VBP) file in a text editor, such as Notepad. You must make the following modification:

1. In the old VBP file, find the following line:

```
Object={0BA686C6-F7D3-101A-993E-0000C0EF6F5E}#1.0#0; THREED32.OCX
```

2. Change the version number to 2.0 as in the following line:

```
Object={0BA686C6-F7D3-101A-993E-0000C0EF6F5E}#2.0#0; THREED20.OCX
```

3. Save the file using your text editor, then open the project as you normally would.

For Visual Basic 5.0

The upgrade from THREED16.OCX or THREED32.OCX is an automatic process. When you open a project that uses a previous version of THREED, you will see a dialog that prompts you to upgrade the project to THREED20.OCX.

Other Upgrade Issues

Previous versions of the SSCheck control found in THREED.VBX, THREED16.OCX and THREED32.OCX supported only two logical states: checked and unchecked. This differs from the standard Visual Basic and Windows check box which supports three states; checked, unchecked and grayed.

If you have existing programs that use a previous version of the SSCheck control, and those programs assume a specific set of values (i.e. that only 2 states are available) you must make sure that the **TripleState** property is set to False. This is the default setting, and ensures full compatibility with previous versions of the control. However, if you wish to use the new capabilities of the control, or use the SSCheck to replace standard check boxes, you should make sure **TripleState** is set to True.

Using ActiveThread Controls

<input checked="" type="radio"/> Left Justified SSOption	<input type="radio"/> Right Justified SSOption
<input type="checkbox"/> Left Justified SSCheck	<input checked="" type="checkbox"/> Right Justified SSCheck

ActiveThread is a set of ActiveX controls designed to add a cutting-edge interface to your applications. ActiveThread consists of seven 32-bit ActiveX controls in two files - THREED20.OCX and SPLITTER.OCX. The THREED20 controls replicate the functions of some of the Windows base controls, adding expanded features to give your applications the active look and feel users have come to expect. Also included in ActiveThread is a Splitter control that provides a flexible, multi-paned container for other controls. Together, the THREED20 controls and Splitter of ActiveThread give you the tools to design attractive and engaging applications, utilizing many of the interface features popularized by the Internet and the World Wide Web.

THREED20.OCX, which contains the THREED20 controls of ActiveThread, is an upgraded version of the THREED.VBX, THREED16.OCX and THREED32.OCX custom controls that were included with the Professional Edition of Microsoft® Visual Basic. It can be substituted directly for older versions of the control without requiring modifications to your program's forms or code. However, you will need to re-compile existing applications if you want them to take advantage of ActiveThread. For more information on upgrading your older THREED applications to ActiveThread, see [Upgrade Notes](#).

ActiveThread provides advanced functionality and performance. It is fully compliant with the ActiveX standard. Its lightweight controls do not require the MFC DLL, making it especially suitable for network or Internet applications. The THREED20 controls give you enhanced interface design features, which you can use to give your application a multimedia look and feel. You can design programs that integrate well with content on the World Wide Web as well as the most recent versions of popular applications suites.

In addition, ActiveThread includes an all-new Splitter control. The Splitter is a container control that is visually and functionally similar to the frames found on many Web sites. It provides a system of resizable panes separated by movable, three-dimensional splitter bars, which you use to group and organize the other controls in your application. The Splitter is packaged as a separate lightweight OCX; you can choose whether or not to include it with the THREED20 controls in your application.

The following is a guide to using the features common to the THREED20 controls. (Most of these will not apply to the SSSplitter.) For information on control-specific features, consult the **Remarks** section of the topic for that particular control, or the topic for the specific property, method, event, object or collection you want to use.

[Using Marquee Captions](#)

[Windowed vs. Windowless Operation](#)

[Control Background Effects](#)

[Transparent Operation](#)

[Adding Pictures to Captions](#)

[Using Mask Colors with Pictures](#)

[Adding Animation To Controls](#)

[Including Sound](#)

[OLE Drag and Drop Operations](#)

[Resizing Panes Through Code \(SSSplitter\)](#)

Using Marquee Captions

Marquee captions are text that incorporates movement. Because movement is attention-getting, marquee captions are useful for drawing attention to a particular part of your application. Marquee captions are also useful when you need to display a long string of text in an area that doesn't have the space to accommodate that text. By using moving text, you can display more information than the area would usually allow.

To create a marquee caption:

1. Enter the text for the **Caption** property of the control as you normally would.
2. Set the **MarqueeStyle** property to select the type of motion effect you want. The choices are None (static), Scrolling, Sliding, Blinking and Bouncing.
3. Set the **MarqueeDirection** property to indicate which direction you would like the marquee text to move. This property is not applicable to the Static or Blinking styles.
4. Set the **MarqueeDelay** property to set the interval of the timer used to control the animation. A higher value will result in slower animation. A very small value (less than 55 for Windows 95, less than 10 for Windows NT) is equivalent to zero. This will cause the control to receive timer messages as fast as the system will allow, which may adversely affect the performance of your application.
5. Set the **MarqueeScrollAmount** to determine the amount of scrolling that will take place with each expiration of the **MarqueeDelay** timer. A higher value will result in faster but choppier scrolling. Lower values produce a smoother, slower effect.
6. If you wish to take some action in response to the behavior of the marquee caption text, use the **MarqueeCycleBegin** event. This event occurs whenever the caption is about to begin a new marquee cycle. What constitutes a cycle varies depending on the **MarqueeStyle**. Generally, it is when the caption has disappeared and is about to reappear, or when the caption changes its movement, such as changing direction.

There is also a **MarqueeCycleEnd** event which occurs after the marquee cycle is complete. This is useful for taking action after a caption has completed an action, such as when text using the Sliding marquee style comes to a halt.

Using Mask Colors with Pictures

Any of the pictures you use with ActiveThread may have transparent areas. You specify which areas of the picture will be transparent through the use of *mask colors*. A mask color is a color that appears in the picture. You specify that this color should be transparent by entering its color value in the **PictureMaskColor** property. You can either enter the value for the property directly, or select the color directly from the color selection dialog of your development environment or the property pages.

Some colors work better than others as mask colors. This is because different video drivers handle the creation of colors in slightly different ways. This may cause a particular mask color to work on one system but not another. To ensure maximum compatibility across systems, the best colors to use as mask colors are primary colors at 100% intensity: red, green or blue. Pure white and black (0% intensity) are also good choices.

Once you specify a mask color using the **PictureMaskColor** property, you must activate color masking for the picture by setting the **PictureUseMask** property to True. This property gives you the ability to select whether or not you want picture masking active for the control.

The **PictureMaskColor** and **PictureUseMask** properties apply to all the pictures specified for the control. They affect the following properties: **Picture**, **PictureBackground**, **PictureDisabled**, **PictureDn**, **PictureDnDisabled**.

You can also use mask colors when specifying custom graphics for the button portion of the SSCheck and SSOOption controls. Because these pictures serve a special purpose, they have their own mask color properties; **CheckBoxMaskColor**, **CheckBoxUseMask**, **OptionBtnMaskColor** and **OptionBtnUseMask**.

To use a mask color with a picture (caption or background):

1. If you wish to use an icon or metafile for your picture, you do not need to take any special action. These formats intrinsically support transparent areas. Any transparent area in the original picture will be transparent when used with an ActiveThread control.
2. If you wish to use a bitmap as your picture, create the bitmap using a single color (such as red) for all the areas of the picture that you wish to be transparent.
3. Specify the bitmap you wish to use by entering its path name for the **Picture** property, either directly or via the File Open dialog.
4. Select the **PictureMaskColor** property, and enter the color you used for the transparent areas of the bitmap. You can enter the color value directly, or choose the color from the color selection dialog / dropdown.
5. Select the **PictureUseMask** property and set its value to True.

Value Property (SSCheck)

[See Also](#)

[Applies To](#)

Description

Returns or sets a value that determines the logical state of the control.

Syntax

object.**Value** [= *integer*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>integer</i>	An integer expression specifying the state of a check box object, as described in Settings.

Settings

The setting of **Value** property for SSSCheck depends on the setting of the **TripleState** property. **TripleState** toggles the control between two or three logical states. The settings for each are as follows:

The settings when *TripleState* is set to FALSE are:

Setting	Description
False	(Default) The check box is not checked.
True	The check box is checked.

The settings when *TripleState* is set to TRUE are:

Setting	Description
0	(Default) The check box is not checked.
1	The check box is checked.
2	The check box is grayed.
> 2 or < 0	The check box is checked.

Remarks

This is the default property of the control.

For backwards compatibility with previous versions of THREEED, the **Value** property of the SSSCheck control returns a Boolean value when **TripleState** is set to False. You can set the **Value** property to a Boolean or an integer value - any non-zero value is treated as True - but the value returned will always be Boolean. When **TripleState** is True, **Value** returns an integer.

Value Property (SSCommand, SSOption, SSRibbon)

[See Also](#)

[Applies To](#)

Description

Returns or sets a value that determines the logical state of the control.

Syntax

object.**Value** [= *boolean*]

Part	Description
-------------	--------------------

<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
---------------	---

<i>boolean</i>	A boolean expression specifying the state of a button object, as described in Settings.
----------------	---

Settings

Setting	Description
----------------	--------------------

True	The button is in a depressed state.
-------------	-------------------------------------

False	(Default) the button is in a raised state.
--------------	--

Remarks

This is the default property of the control.

When the value of an SSOption button is set to True, the values of all the other option buttons in the same container are automatically set to False.

You can simulate a button click by setting the value of a SSCommand or SSRibbon to True. This has the same effect as invoking the **DoClick** method for the control.

The **Value** property of the SSCommand button is available only at run-time.

Width Property

[See Also](#) [Applies To](#)

Description

Returns or sets the horizontal dimension of an object.

Syntax

object.**Width** [= *number*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>number</i>	An single precision expression specifying the width of the object. The value returned is in the scale units of the object's container.

Remarks

This is an [extender](#) property, except when applied to the **Pane** object..

Use the **Height**, **Width**, **Left**, and **Top** properties for operations or calculations based on an object's total area, such as sizing or moving the object.

Windowed vs. Windowless Operation

Most development platforms support *windowed* controls. This means that each control on a form exists as a separate window, has its own window handle, and consumes resources which are necessary for the "housekeeping" of opening and keeping track of windows. The window requirements of custom controls place a limit on the number of controls you can have in an application without adversely affecting overall program and system performance.

Some development environments provide support for *windowless* controls. Windowless controls are lightweight; they do not have a window handle and do not consume system resources in the same way (or at the same rate) as windowed controls. Typically, when using windowless controls, you can use more controls in your application without affecting its efficiency, and the application as a whole should be leaner, with a reduced "memory footprint."

ActiveThread's non-container controls (SSCheck, SSCommand, SSOption & SSRibbon) can operate in windowless mode in environments that support that type of operation. However, they can also operate in standard windowed mode. Container controls (SSFrame, SSPanel, SSSplitter) always operate in windowed mode. Windowless controls are supported through the use of the *IWindowlessControlSite* interface; any environment that uses this interface will support ActiveThread controls in windowless mode.

How the controls operate is completely transparent to the programmer. If your environment supports windowless controls, the ActiveThread controls will take advantage of windowless operation until you need to use some "window-specific" feature of the control, such as obtaining its window handle. The controls can dynamically readjust their mode of operation to assure maximum flexibility and efficiency.

Windowless Property

[See Also](#)

[Applies To](#)

Description

Returns or sets a value that determines whether the control can operate in windowless mode.

Syntax

object.**Windowless** [= *boolean*]

Part	Description
<i>object</i>	An object expression that evaluates to an object or a control in the Applies To list.
<i>boolean</i>	A boolean expression specifying the window mode capabilities of the control, as described in Settings.

Settings

Setting	Description
True	(Default) The control will operate in windowless mode if it supported by the environment.
False	The control will not operate in windowless mode.

Remarks

This property is available at design time only.

The non-container ActiveThreed controls can operate in windowless mode in environments that support it. If you do not wish to use windowless mode, you can set this property to False, which will force the controls into windowed mode at all times. You may want to do this if your operating environment does not properly support the controls in windowless mode.

(About) Property applies to:

SSCheck control

SSCommand control

SSFrame control

SSOption control

SSPanel control

SSRibbon control

SSSplitter control

(Custom) Property applies to:

SSCheck control

SSCommand control

SSFrame control

SSOption control

SSPanel control

SSRibbon control

SSSplitter control

Add Method applies to:

Panes collection

Align Property applies to:

SSPanel control

SSSplitter control

Alignment Property applies to:

SSCheck control

SSCommand control

SSFrame control

SSOption control

SSPanel control

SSRibbon control

SSSplitter control

AutoRepeat Property applies to:

SSCommand control

AutoSize Property (SSCommand & SSRibbon) applies to:

SSCommand control

SSRibbon control

AutoSize Property (SSPanel) applies to:

SSPanel control

AutoSize Property (SSplitter) applies to:

SSplitter control

BackColor Property applies to:

SSCheck control

SSCommand control

SSFrame control

SSOption control

SSPanel control

SSRibbon control

SSSplitter control

BackStyle Property applies to:

SSCheck control

SSCommand control

SSFrame control

SSOption control

SSPanel control

SSRibbon control

BevellInner Property applies to:

SSPanel control

BevelOuter Property applies to:

SSPanel control

BevelWidth Property applies to:

SSCommand control

SSPanel control

SSRibbon control

Bold Property applies to:

Font object

BorderStyle Property applies to:

SSplitter control

BorderWidth Property applies to:

SSPanel control

ButtonStyle Property applies to:

SSCommand control

SSRibbon control

Cancel Property applies to:

SSCommand control

Caption Property applies to:

SSCheck control

SSCommand control

SSFrame control

SSOption control

SSPanel control

SSRibbon control

CaptionStyle Property applies to:

SSCheck control

SSCommand control

SSFrame control

SSOption control

SSPanel control

SSRibbon control

CheckBoxGraphics Property applies to:

SSCheck control

CheckBoxMaskColor Property applies to:

SSCheck control

CheckBoxUseMask Property applies to:

SSCheck control

Clear Method applies to:

ssDataObject object

ssDataObjectFiles collection

Click Event applies to:

SSCheck control

SSCommand control

SSOption control

SSPanel control

SSRibbon control

SSSplitter control

ClipControls Property applies to:

SSFrame control

SSPanel control

SSplitter control

Control Property applies to:

Pane object

ControlName Property applies to:

Pane object

Count Property applies to:

Panes collection

ssDataObjectFiles collection

DataField Property applies to:

SSCheck control

SSPanel control

DataSource Property applies to:

SSCheck control

SSPanel control

DbClick Event applies to:

SSFrame control

SSPanel control

SSplitter control

Default Property applies to:

SSCommand control

DoClick Method applies to:

SSCommand control

DragMode Property applies to:

SSCheck control

SSCommand control

SSFrame control

SSOption control

SSPanel control

SSRibbon control

SSSplitter control

Enabled Property applies to:

SSCheck control

SSCommand control

SSFrame control

SSOption control

SSPanel control

SSRibbon control

SSSplitter control

Files Property applies to:

ssDataObject object

FloodColor Property applies to:

SSPanel control

FloodFillStyle Property applies to:

SSPanel control

FloodPercent Property applies to:

SSPanel control

FloodShowPct Property applies to:

SSPanel control

FloodType Property applies to:

SSPanel control

Font Object applies to:

SSCheck control

SSCommand control

SSFrame control

SSOption control

SSPanel control

SSRibbon control

Font Property applies to:

SSCheck control

SSCommand control

SSFrame control

SSOption control

SSPanel control

SSRibbon control

Font3D Property applies to:

SSCheck control

SSCommand control

SSFrame control

SSOption control

SSPanel control

SSRibbon control

ForeColor Property applies to:

SSCheck control

SSCommand control

SSFrame control

SSOption control

SSPanel control

SSRibbon control

GetData Method applies to:

ssDataObject object

GetFormat Method applies to:

ssDataObject object

GroupAllowAllUp Property applies to:

SSRibbon control

GroupNumber Property applies to:

SSRibbon control

Height Property applies to:

SSCheck control

SSCommand control

SSFrame control

SSOption control

SSPanel control

SSRibbon control

SSSplitter control

Pane object

HelpContextID Property applies to:

SSCheck control

SSCommand control

SSFrame control

SSOption control

SSPanel control

SSRibbon control

Hwnd Property applies to:

SSCheck control

SSCommand control

SSFrame control

SSOption control

SSPanel control

SSRibbon control

SSSplitter control

Italic Property applies to:

SSCheck control

SSCommand control

SSFrame control

SSOption control

SSPanel control

SSRibbon control

Item Method applies to:

Panes collection

ssDataObjectFiles collection

KeyDown Event applies to:

SSCheck control

SSCommand control

SSOption control

KeyPress Event applies to:

SSCheck control

SSCommand control

SSOption control

KeyUp Event applies to:

SSCheck control

SSCommand control

SSOption control

Left Property applies to:

SSCheck control

SSCommand control

SSFrame control

SSOption control

SSPanel control

SSRibbon control

SSSplitter control

Pane object

LockHeight Property applies to:

Pane object

LockWidth Property applies to:

Pane object

Locked Property applies to:

SSplitter control

MarqueeCycleBegin Event applies to:

SSCheck control

SSCommand control

SSFrame control

SSOption control

SSPanel control

SSRibbon control

MarqueeCycleEnd Event applies to:

SSCheck control

SSCommand control

SSFrame control

SSOption control

SSPanel control

SSRibbon control

MarqueeDelay Property applies to:

SSCheck control

SSCommand control

SSFrame control

SSOption control

SSPanel control

SSRibbon control

MarqueeDirection Property applies to:

SSCheck control

SSCommand control

SSFrame control

SSOption control

SSPanel control

SSRibbon control

MarqueeScrollAmount Property applies to:

SSCheck control

SSCommand control

SSFrame control

SSOption control

SSPanel control

SSRibbon control

MarqueeStyle Property applies to:

SSCheck control

SSCommand control

SSFrame control

SSOption control

SSPanel control

SSRibbon control

MinHeight Property applies to:

Pane object

MinWidth Property applies to:

Pane object

MouseDown Event applies to:

SSCheck control

SSCommand control

SSFrame control

SSOption control

SSPanel control

SSRibbon control

MouseEnter Event applies to:

SSCheck control

SSCommand control

SSFrame control

SSOption control

SSPanel control

SSRibbon control

MouseExit Event applies to:

SSCheck control

SSCommand control

SSFrame control

SSOption control

SSPanel control

SSRibbon control

Mouselcon Property applies to:

SSCheck control

SSCommand control

SSFrame control

SSOption control

SSPanel control

SSRibbon control

MouseMove Event applies to:

SSCheck control

SSCommand control

SSFrame control

SSOption control

SSPanel control

SSRibbon control

MousePointer Property applies to:

SSCheck control

SSCommand control

SSFrame control

SSOption control

SSPanel control

SSRibbon control

MouseUp Event applies to:

SSCheck control

SSCommand control

SSFrame control

SSOption control

SSPanel control

SSRibbon control

Name Property applies to:

SSCheck control

SSCommand control

SSFrame control

SSOption control

SSPanel control

SSRibbon control

SSSplitter control

Font object

Pane object

Name Property (Font object) applies to:

Font object

OLECompleteDrag Event applies to:

SSCheck control

SSCommand control

SSFrame control

SSOption control

SSPanel control

SSRibbon control

OLEDrag Method applies to:

SSCheck control

SSCommand control

SSFrame control

SSOption control

SSPanel control

SSRibbon control

SSSplitter control

OLEDragDrop Event applies to:

SSCheck control

SSCommand control

SSFrame control

SSOption control

SSPanel control

SSRibbon control

OLEDragOver Event applies to:

SSCheck control

SSCommand control

SSFrame control

SSOption control

SSPanel control

SSRibbon control

OLEDropMode Property applies to:

SSCheck control

SSCommand control

SSFrame control

SSOption control

SSPanel control

SSRibbon control

OLEGiveFeedback Event applies to:

SSCheck control

SSCommand control

SSFrame control

SSOption control

SSPanel control

SSRibbon control

OLESetData Event applies to:

SSCheck control

SSCommand control

SSFrame control

SSOption control

SSPanel control

SSRibbon control

OLEStartDrag Event applies to:

SSCheck control

SSCommand control

SSFrame control

SSOption control

SSPanel control

SSRibbon control

OptionBtnGraphics Property applies to:

SSOption control

OptionBtnMaskColor Property applies to:

SSOption control

OptionBtnUseMask Property applies to:

SSOption control

Outline Property applies to:

SSCommand control

SSPanel control

SSRibbon control

Pane Object applies to:

SSplitter control

PaneFromControl Method applies to:

SSplitter control

PaneFromPosition Method applies to:

SSplitter control

PaneFromPositionEx Method applies to:

SSplitter control

Panes Collection applies to:

SSplitter control

Picture Property applies to:

SSCheck control

SSCommand control

SSFrame control

SSOption control

SSPanel control

SSRibbon control

PictureAlignment Property applies to:

SSCheck control

SSCommand control

SSFrame control

SSOption control

SSPanel control

SSRibbon control

PictureAnimationDelay Property applies to:

SSCheck control

SSCommand control

SSFrame control

SSOption control

SSPanel control

SSRibbon control

PictureAnimationEnabled Property applies to:

SSCheck control

SSCommand control

SSFrame control

SSOption control

SSPanel control

SSRibbon control

PictureBackground Property applies to:

SSFrame control

SSPanel control

PictureBackgroundStyle Property applies to:

SSFrame control

SSPanel control

PictureDisabled Property applies to:

SSCommand control

SSRibbon control

PictureDisabledFrames Property applies to:

SSCommand control

SSRibbon control

PictureDn Property applies to:

SSCommand control

SSRibbon control

PictureDnChange Property applies to:

SSRibbon control

PictureDnDisabled Property applies to:

SSRibbon_control

PictureDnDisabledFrames Property applies to:

SSRibbon control

PictureDnFrames Property applies to:

SSCommand control

SSRibbon control

PictureFrames Property applies to:

SSCheck control

SSCommand control

SSFrame control

SSOption control

SSPanel control

SSRibbon control

PictureMaskColor Property applies to:

SSCheck control

SSCommand control

SSFrame control

SSOption control

SSPanel control

SSRibbon control

PictureUseMask Property applies to:

SSCheck control

SSCommand control

SSFrame control

SSOption control

SSPanel control

SSRibbon control

PlaySoundFile Method applies to:

SSCheck control

SSCommand control

SSFrame control

SSOption control

SSPanel control

SSRibbon control

SSSplitter control

Refresh Method applies to:

SSCheck control

SSCommand control

SSFrame control

SSOption control

SSPanel control

SSRibbon control

SSSplitter control

Remove Method applies to:

Panes collection

ssDataObjectFiles collection

Resize Event applies to:

SSplitter control

RoundedCorners Property applies to:

SSCommand control

SSPanel control

SSRibbon control

SetData Method applies to:

ssDataObject object

SetFocus Method applies to:

SSCheck control

SSCommand control

SSOption control

ShadowStyle Property applies to:

SSFrame control

Size Property applies to:

Font object

SplitterBarAppearance Property applies to:

SSplitter control

SplitterBarJoinStyle Property applies to:

SSSplitter control

SplitterBarWidth Property applies to:

SSSplitter control

SplitterEndDrag Property applies to:

SSplitter control

SplitterResizeStyle Property applies to:

SSplitter control

SplitterStartDrag Event applies to:

SSplitter control

Strikethrough Property applies to:

Font object

TagVariant Property applies to:

SSCheck control

SSCommand control

SSFrame control

SSOption control

SSPanel control

SSRibbon control

SSSplitter control

Top Property applies to:

SSCheck control

SSCommand control

SSFrame control

SSOption control

SSPanel control

SSRibbon control

SSSplitter control

Pane object

TriState Property applies to:

SSCheck control

Underline Property applies to:

Font object

Value Property (SSCheck) applies to:

SSCheck control

Value Property (SSCommand, SSOption, SSRibbon) apply to:

SSCommand control

SSOption control

SSRibbon control

Width Property applies to:

SSCheck control

SSCommand control

SSFrame control

SSOption control

SSPanel control

SSRibbon control

SSSplitter control

Pane object

Width Property applies to:

SSCheck control

SSCommand control

SSOption control

SSRibbon control

SSSplitter control

ssDataObject Object applies to:

SSCheck control

SSCommand control

SSFrame control

SSOption control

SSPanel control

SSRibbon control

ssDataObjectFiles Collection applies to:

SSCheck control

SSCommand control

SSFrame control

SSOption control

SSPanel control

SSRibbon control

Add Method Example

The following code adds 1 vertical and 1 horizontal pane to the Splitter control.

```
SSSplitter1.Panes.Add "Pane A", 2    'split vertically  
SSSplitter1.Panes.Add "Pane B", 1    'split horizontally
```


Align Property Example

The following code aligns the Splitter control to the top of its container.

```
SSSplitter1.Align = 1
```

Alignment Property Example

The following code aligns the caption of an `SSCommand` button control to the bottom-center of the control.

```
SSCommand1.Alignment = 8
```

AutoRepeat Property Example

This example increments the numeric value in a label control as an SSCommand button is pressed and held down:

Place a label control and an SSCommand button on a form. Then place the following code in the **Load** event of the form:

```
Private Sub Form_Load()  
    Label1.Caption = 1  
    SSCommand1.Caption = "Increase!"  
    SSCommand1.AutoRepeat = True  
End Sub
```

Place the following code into the **Click** event of the SSCommand button:

```
Private Sub SSCommand1_Click()  
    Label1.Caption = Str$(Val(Label1.Caption) + 1)  
End Sub
```

Now run the program and hold down the SSCommand button.

Control Property Example

Assuming a picture control (Picture1) is placed within Panes(0) of an SSSplitter control, the following code will set the Picture property of another control (namely Picture2) to the picture in the Picture1 control via the **Control** property.

```
Private Sub Command1_Click()  
    Picture1.Picture = "C:\MYPIC.BMP"  
    Picture2.Picture = SSSplitter1.Panes(0).Control.Picture  
End Sub
```

ControlName Property Example

Assuming a picture control (Picture1) is placed within Pane(0) of an SSSplitter control, the following code will display a message box containing the name of the picture control.

```
Private Sub Command1_Click()  
    MsgBox SSSplitter1.Panes(0).ControlName  
End Sub
```

DoClick Method Example

The following code triggers an SSCommand button's **Click** event when a regular command button is pressed.

Place the following code in the **Click** event of a command button.

```
Private Sub Command1_Click()  
    SSCommand1.DoClick  
End Sub
```

Place the following code in the **Click** event of the SSCommand button.

```
Private Sub SSCommand1_Click()  
    MsgBox "SSCommand1's Click event was fired!"  
End Sub
```

Now run the program and press the regular command button (Command1). You will notice that the message placed in the **Click** event of the SSCommand button is displayed.

FloodColor Property Example

This example increments and decrements a progress bar created using an SSPanel control.

Place two command buttons (Command1 and Command2), and an SSPanel control on a form. Then place the following code in the **Load** event of the form:

```
Private Sub Form_Load()  
    'Size/Position window and controls  
    Me.Width = 3400  
    Me.Height = 3500  
    SSPanel1.Move 100, 100, 1000, 2500  
    Command1.Move 1380, 1440, 1445, 495  
    Command2.Move 1380, 2160, 1445, 495  
  
    'Initialize properties  
    Command1.Caption = "Increment"  
    Command2.Caption = "Decrement"  
  
    SSPanel1.BevelOuter = 1  
    SSPanel1.FloodType = 4  
    SSPanel1.FloodFillStyle = 1  
    SSPanel1.FloodColor = RGB(0, 255, 0)  
    SSPanel1.FloodPercent = 0  
    SSPanel1.FloodShowPct = True  
End Sub
```

Place the following code in the **Click** event of the Command1 button:

```
Private Sub Command1_Click()  
    If SSPanel1.FloodPercent < 100 Then  
        SSPanel1.FloodPercent = SSPanel1.FloodPercent + 10  
    End If  
End Sub
```

Place the following code in the **Click** event of the Command2 button:

```
Private Sub Command2_Click()  
    If SSPanel1.FloodPercent > 0 Then  
        SSPanel1.FloodPercent = SSPanel1.FloodPercent - 10  
    End If  
End Sub
```

Now run the program and repeatedly press the Increment and Decrement buttons. You will notice that a progress bar of the style, flood fill style and color you have specified is drawn in the SSPanel as you press the buttons. In addition, the percentage of fill is also displayed in the SSPanel.

FloodFillStyle Property Example

This example increments and decrements a progress bar created using an SSPanel control.

Place two command buttons (Command1 and Command2), and an SSPanel control on a form. Then place the following code in the **Load** event of the form:

```
Private Sub Form_Load()  
    'Size/Position window and controls  
    Me.Width = 3400  
    Me.Height = 3500  
    SSPanel1.Move 100, 100, 1000, 2500  
    Command1.Move 1380, 1440, 1445, 495  
    Command2.Move 1380, 2160, 1445, 495  
  
    'Initialize properties  
    Command1.Caption = "Increment"  
    Command2.Caption = "Decrement"  
  
    SSPanel1.BevelOuter = 1  
    SSPanel1.FloodType = 4  
    SSPanel1.FloodFillStyle = 1  
    SSPanel1.FloodColor = RGB(0, 255, 0)  
    SSPanel1.FloodPercent = 0  
    SSPanel1.FloodShowPct = True  
End Sub
```

Place the following code in the **Click** event of the Command1 button:

```
Private Sub Command1_Click()  
    If SSPanel1.FloodPercent < 100 Then  
        SSPanel1.FloodPercent = SSPanel1.FloodPercent + 10  
    End If  
End Sub
```

Place the following code in the **Click** event of the Command2 button:

```
Private Sub Command2_Click()  
    If SSPanel1.FloodPercent > 0 Then  
        SSPanel1.FloodPercent = SSPanel1.FloodPercent - 10  
    End If  
End Sub
```

Now run the program and repeatedly press the Increment and Decrement buttons. You will notice that a progress bar of the style, FloodFill style and color you have specified is drawn in the SSPanel as you press the buttons. In addition, the percentage of fill is also displayed in the SSPanel.

FloodPercent Property Example

This example increments and decrements a progress bar created using an SSPanel control.

Place two command buttons (Command1 and Command2), and an SSPanel control on a form. Then place the following code in the **Load** event of the form:

```
Private Sub Form_Load()  
    'Size/Position window and controls  
    Me.Width = 3400  
    Me.Height = 3500  
    SSPanel1.Move 100, 100, 1000, 2500  
    Command1.Move 1380, 1440, 1445, 495  
    Command2.Move 1380, 2160, 1445, 495  
  
    'Initialize properties  
    Command1.Caption = "Increment"  
    Command2.Caption = "Decrement"  
  
    SSPanel1.BevelOuter = 1  
    SSPanel1.FloodType = 4  
    SSPanel1.FloodFillStyle = 1  
    SSPanel1.FloodColor = RGB(0, 255, 0)  
    SSPanel1.FloodPercent = 0  
    SSPanel1.FloodShowPct = True  
End Sub
```

Place the following code in the **Click** event of the Command1 button:

```
Private Sub Command1_Click()  
    If SSPanel1.FloodPercent < 100 Then  
        SSPanel1.FloodPercent = SSPanel1.FloodPercent + 10  
    End If  
End Sub
```

Place the following code in the **Click** event of the Command2 button:

```
Private Sub Command2_Click()  
    If SSPanel1.FloodPercent > 0 Then  
        SSPanel1.FloodPercent = SSPanel1.FloodPercent - 10  
    End If  
End Sub
```

Now run the program and repeatedly press the Increment and Decrement buttons. You will notice that a progress bar of the style, floodfill style and color you have specified is drawn in the SSPanel as you press the buttons. In addition, the percentage of fill is also displayed in the SSPanel.

FloodShowPct Property Example

This example increments and decrements a progress bar created using an SSPanel control.

Place two command buttons (Command1 and Command2), and an SSPanel control on a form. Then place the following code in the **Load** event of the form:

```
Private Sub Form_Load()  
    'Size/Position window and controls  
    Me.Width = 3400  
    Me.Height = 3500  
    SSPanel1.Move 100, 100, 1000, 2500  
    Command1.Move 1380, 1440, 1445, 495  
    Command2.Move 1380, 2160, 1445, 495  
  
    'Initialize properties  
    Command1.Caption = "Increment"  
    Command2.Caption = "Decrement"  
  
    SSPanel1.BevelOuter = 1  
    SSPanel1.FloodType = 4  
    SSPanel1.FloodFillStyle = 1  
    SSPanel1.FloodColor = RGB(0, 255, 0)  
    SSPanel1.FloodPercent = 0  
    SSPanel1.FloodShowPct = True  
End Sub
```

Place the following code in the **Click** event of the Command1 button:

```
Private Sub Command1_Click()  
    If SSPanel1.FloodPercent < 100 Then  
        SSPanel1.FloodPercent = SSPanel1.FloodPercent + 10  
    End If  
End Sub
```

Place the following code in the **Click** event of the Command2 button:

```
Private Sub Command2_Click()  
    If SSPanel1.FloodPercent > 0 Then  
        SSPanel1.FloodPercent = SSPanel1.FloodPercent - 10  
    End If  
End Sub
```

Now run the program and repeatedly press the Increment and Decrement buttons. You will notice that a progress bar of the style, flood fill style and color you have specified is drawn in the SSPanel as you press the buttons. In addition, the percentage of fill is also displayed in the SSPanel.

FloodType Property Example

This example increments and decrements a progress bar created using an SSPanel control.

Place two command buttons (Command1 and Command2), and an SSPanel control on a form. Then place the following code in the **Load** event of the form:

```
Private Sub Form_Load()  
    'Size/Position window and controls  
    Me.Width = 3400  
    Me.Height = 3500  
    SSPanel1.Move 100, 100, 1000, 2500  
    Command1.Move 1380, 1440, 1445, 495  
    Command2.Move 1380, 2160, 1445, 495  
  
    'Initialize properties  
    Command1.Caption = "Increment"  
    Command2.Caption = "Decrement"  
  
    SSPanel1.BevelOuter = 1  
    SSPanel1.FloodType = 4  
    SSPanel1.FloodFillStyle = 1  
    SSPanel1.FloodColor = RGB(0, 255, 0)  
    SSPanel1.FloodPercent = 0  
    SSPanel1.FloodShowPct = True  
End Sub
```

Place the following code in the **Click** event of the Command1 button:

```
Private Sub Command1_Click()  
    If SSPanel1.FloodPercent < 100 Then  
        SSPanel1.FloodPercent = SSPanel1.FloodPercent + 10  
    End If  
End Sub
```

Place the following code in the **Click** event of the Command2 button:

```
Private Sub Command2_Click()  
    If SSPanel1.FloodPercent > 0 Then  
        SSPanel1.FloodPercent = SSPanel1.FloodPercent - 10  
    End If  
End Sub
```

Now run the program and repeatedly press the Increment and Decrement buttons. You will notice that a progress bar of the style, floodfill style and color you have specified is drawn in the SSPanel as you press the buttons. In addition, the percentage of fill is also displayed in the SSPanel.

Item Method Example

Assuming a picture control (Picture1) is placed within Pane(0) of an SSSplitter control, the following code will display a message box containing the name of the picture control.

```
MsgBox SSSplitter1.Panes.Item(0).ControlName
```

Since Item is the default method of the Panes collection, this code is equivalent to:

```
MsgBox SSSplitter1.Panes(0).ControlName
```

MarqueeCycleBegin Event Example

This example randomly sets the color of the SSCommand button's caption each time a marquee cycle is completed.

Place the following code in the **Load** event of the form:

```
Private Sub Form_Load()  
    Randomize  
    SSCommand1.Caption = "Hello!"  
    SSCommand1.Font.Name = "Arial"  
    SSCommand1.Font.Bold = True  
    SSCommand1.Font.Size = 18  
    SSCommand1.Move 1000, 1000, 2500, 1000  
  
    SSCommand1.MarqueeStyle = 4 'Bouncing text  
    SSCommand1.MarqueeDirection = ssMDLeftToRight  
    SSCommand1.MarqueeDelay = 0  
End Sub
```

Place the following code in the **MarqueeCycleBegin** event of the SSCommand button.

```
Private Sub SSCommand1_MarqueeCycleBegin()  
    SSCommand1.ForeColor = RGB(Int(Rnd * 255) + 1, Int(Rnd * 255) + 1,  
    Int(Rnd * 255) + 1)  
End Sub
```

MarqueeDelay Property Example

This example slows down the scrolling speed of the marquee caption in an SSCommand button each time the button is clicked.

Place the following code in the **Click** event of the form:

```
Private Sub SSCommand1_Click()  
    SSCommand1.MarqueeDelay = SSCommand1.MarqueeDelay + 30  
End Sub
```

MarqueeDirection Property Example

This example randomly sets the color of the SSCommand button's caption each time a marquee cycle is completed.

Place the following code in the **Load** event of the form:

```
Private Sub Form_Load()  
    Randomize  
    SSCommand1.Caption = "Hello!"  
    SSCommand1.Font.Name = "Arial"  
    SSCommand1.Font.Bold = True  
    SSCommand1.Font.Size = 18  
    SSCommand1.Move 1000, 1000, 2500, 1000  
  
    SSCommand1.MarqueeStyle = 4 'Bouncing text  
    SSCommand1.MarqueeDirection = ssMDLeftToRight  
    SSCommand1.MarqueeDelay = 0  
End Sub
```

Place the following code in the **MarqueeCycleBegin** event of the SSCommand button.

```
Private Sub SSCommand1_MarqueeCycleBegin()  
    SSCommand1.ForeColor = RGB(Int(Rnd * 255) + 1, Int(Rnd * 255) + 1,  
    Int(Rnd * 255) + 1)  
End Sub
```

MarqueeStyle Property Example

This example randomly sets the color of the SSCommand button's caption each time a marquee cycle is completed.

Place the following code in the **Load** event of the form:

```
Private Sub Form_Load()  
    Randomize  
    SSCommand1.Caption = "Hello!"  
    SSCommand1.Font.Name = "Arial"  
    SSCommand1.Font.Bold = True  
    SSCommand1.Font.Size = 18  
    SSCommand1.Move 1000, 1000, 2500, 1000  
  
    SSCommand1.MarqueeStyle = 4 'Bouncing text  
    SSCommand1.MarqueeDirection = ssMDLeftToRight  
    SSCommand1.MarqueeDelay = 0  
End Sub
```

Place the following code in the **MarqueeCycleBegin** event of the SSCommand button.

```
Private Sub SSCommand1_MarqueeCycleBegin()  
    SSCommand1.ForeColor = RGB(Int(Rnd * 255) + 1, Int(Rnd * 255) + 1,  
    Int(Rnd * 255) + 1)  
End Sub
```


MouseEnter Event Example

The following code increases the animation speed of an SSCommand button as the mouse is moved over it:

Place an SSCommand button on a form. Then place the following code into the **Load** event of the form:

```
Private Sub Form_Load()  
    SSCommand1.Width = 2000  
    SSCommand1.Height = 1500  
    SSCommand1.Caption = "Exit"  
    SSCommand1.Alignment = ssCenterBottom  
    SSCommand1.Picture = LoadPicture("C:\Sheridan\ActiveThreed\Graphics\  
Animations\Segmented Bitmaps\Red light(16).bmp")  
    SSCommand1.PictureMaskColor = vbBlue  
    SSCommand1.PictureAnimationDelay = 100  
    SSCommand1.PictureFrames = 16  
End Sub
```

Place the following code into the **MouseEnter** event of the SSCommand button:

```
Private Sub SSCommand1_MouseEnter(ByVal Button As Integer, ByVal Shift  
As Integer, ByVal X As Single, ByVal Y As Single)  
    SSCommand1.PictureAnimationDelay = 10  
End Sub
```

Place the following code into the **MouseExit** event of the SSCommand button:

```
Private Sub SSCommand1_MouseExit(ByVal Button As Integer, ByVal Shift As  
Integer, ByVal X As Single, ByVal Y As Single)  
    SSCommand1.PictureAnimationDelay = 100  
End Sub
```

Now run the program and move the mouse pointer over the command button. You will notice that the animation speeds up as you pass the mouse over the control.

MouseExit Event Example

The following code increases the animation speed of an SSCommand button as the mouse is moved over it:

Place an SSCommand button on a form. Then place the following code into the **Load** event of the form:

```
Private Sub Form_Load()  
    SSCommand1.Width = 2000  
    SSCommand1.Height = 1500  
    SSCommand1.Caption = "Exit"  
    SSCommand1.Alignment = ssCenterBottom  
    SSCommand1.Picture = LoadPicture("C:\Sheridan\ActiveThreed\Graphics\  
Animations\Segmented Bitmaps\Red light(16).bmp")  
    SSCommand1.PictureMaskColor = vbBlue  
    SSCommand1.PictureAnimationDelay = 100  
    SSCommand1.PictureFrames = 16  
End Sub
```

Place the following code into the **MouseEnter** event of the SSCommand button:

```
Private Sub SSCommand1_MouseEnter(ByVal Button As Integer, ByVal Shift  
As Integer, ByVal X As Single, ByVal Y As Single)  
    SSCommand1.PictureAnimationDelay = 10  
End Sub
```

Place the following code into the **MouseExit** event of the SSCommand button:

```
Private Sub SSCommand1_MouseExit(ByVal Button As Integer, ByVal Shift As  
Integer, ByVal X As Single, ByVal Y As Single)  
    SSCommand1.PictureAnimationDelay = 100  
End Sub
```

MouseMove Event Example

The following example alternates the direction in which a marquee is scrolling depending on the position of the mouse pointer over the control.

Place the following code into the **Load** event of the form:

```
Private Sub Form_Load()  
    SSPanel1.MarqueeStyle = 1  
    SSPanel1.MarqueeDelay = 1  
    SSPanel1.MarqueeDirection = ssMDLeftToRight  
    SSPanel1.Caption = "Right -->"  
End Sub
```

Add the following code to the **MouseMove** event of the panel"

```
Private Sub SSPanel1_MouseMove(Button As Integer, Shift As Integer, X As  
Single, Y As Single)  
    If X < SSPanel1.Width / 2 Then  
        SSPanel1.MarqueeDirection = ssMDRightToLeft  
        SSPanel1.Caption = "<-- Left"  
    Else  
        SSPanel1.MarqueeDirection = ssMDLeftToRight  
        SSPanel1.Caption = "Right -->"  
    End If  
End Sub
```

Now run the program and move the mouse pointer over the command button. You will notice that when the mouse is over the left half of the SSPanel, the marquee scrolls from right to left. When the mouse is over the right half of the SSPanel, the marquee scrolls from left to right.

OptionBtnGraphics Property Example

The following code sets an option button graphic, and graphic mask color. It then applies the mask color to make all pure-blue areas in the graphic transparent.

```
SSOption1.OptionBtnGraphics = LoadPicture("MYOPTIONBTNPIC.BMP")
SSOption1.OptionBtnMaskColor = vbBlue
SSOption1.OptionBtnUseMask = True
```

OptionBtnMaskColor Property Example

The following code sets an option button graphic, and graphic mask color. It then applies the mask color to make all pure-blue areas in the graphic transparent.

```
SSOption1.OptionBtnGraphics = LoadPicture("MYOPTIONBTNPIC.BMP")  
SSOption1.OptionBtnMaskColor = vbBlue  
SSOption1.OptionBtnUseMask = True
```

OptionBtnUseMask Property Example

The following code sets an option button graphic, and graphic mask color. It then applies the mask color to make all pure-blue areas in the graphic transparent.

```
SSOption1.OptionBtnGraphics = LoadPicture("MYOPTIONBTNPIC.BMP")  
SSOption1.OptionBtnMaskColor = vbBlue  
SSOption1.OptionBtnUseMask = True
```

Pane Object Example

The following code adds a 3 new **Pane** objects to the **Panes** collection, for a total of 4 panes:

```
SSSplitter1.Panes.Add "Pane A", 2    'split vertically
SSSplitter1.Panes.Add "Pane B", 1    'split horizontally
SSSplitter1.Panes.Add "Pane C", 1    'split horizontally

MsgBox SSSplitter1.Panes.Count'display number of panes
```

PaneFromControl Method Example

Assuming four Pane objects are created and a picture box (Picture1) is placed in Pane D, the following code will display the name of the pane that contains the picture box:

```
Dim MyPane As Pane

Set MyPane = SSSplitter1.PaneFromControl(Picture1)
MsgBox MyPane.Name
```


PaneFromPosition Method Example

This example creates a 4-way split window using the SSSplitter control and highlights each pane as the mouse pointer passes over it. First, place an SSSplitter control and a picture box on a form.

Place the following code into the **Load** event of the form:

```
Private Sub Form_Load()  
    SSSplitter1.Panes.Add "Pane A", 2  
    SSSplitter1.Panes.Add "Pane B", 1  
    SSSplitter1.Panes.Add "Pane C", 1  
    SSSplitter1.AutoSize = ssAutoSizeFillContainer  
  
    Picture1.Visible = False  
    Picture1.BackColor = vbBlue  
End Sub
```

Place the following code into the **MouseMove** event of the SSSplitter control:

```
Private Sub SSSplitter1_MouseMove(Button As Integer, Shift As Integer, x  
As Single, y As Single)  
    Dim MyPane As Pane  
  
    If Picture1.Visible = False Then Picture1.Visible = True  
  
    On Error GoTo ErrorHandler  
    Set MyPane = SSSplitter1.PaneFromPosition(x, y)  
    MyPane.Control = Picture1  
  
    Exit Sub  
  
ErrorHandler:  
    ' Used to handle the movement of the mouse pointer over  
    ' a splitter bar  
End Sub
```

Now run the program and move the mouse pointer over each of the 4 panes of the window. The **PaneFromPosition** method was first used to obtain the Pane object for the pane the mouse pointer was over. Next, the **Control** property of the Pane object was set to the picture box. This makes the picture box a child of the pane.

PaneFromPositionEx Method Example

This example creates a 4-way split window using the SSSplitter control and highlights each pane as the mouse pointer passes over it. First, place an SSSplitter control and a picture box on a form.

Place the following code into the **Load** event of the form:

```
Private Sub Form_Load()  
    SSSplitter1.Panes.Add "Pane A", 2  
    SSSplitter1.Panes.Add "Pane B", 1  
    SSSplitter1.Panes.Add "Pane C", 1  
    SSSplitter1.AutoSize = ssAutoSizeFillContainer  
  
    Picture1.Visible = False  
    Picture1.BackColor = vbBlue  
End Sub
```

Place the following code into the **MouseMove** event of the SSSplitter control:

```
Private Sub SSSplitter1_MouseMove(Button As Integer, Shift As Integer, x  
As Single, y As Single)  
    Dim MyPane As Pane  
  
    If Picture1.Visible = False Then Picture1.Visible = True  
  
    Set MyPane = SSSplitter1.PaneFromPositionEx(x, y)  
  
    If MyPane Is Nothing Then  
        ' Take no action  
    Else  
        MyPane.Control = Picture1  
    EndIf  
  
End Sub
```

Now run the program and move the mouse pointer over each of the 4 panes of the window. The **PaneFromPositionEx** method was first used to obtain the **Pane** object for the pane the mouse pointer was over. Next, the **Control** property of the **Pane** object was set to the picture box. This makes the picture box a child of the pane.

Panes Collection Example

The following code adds a 3 new Pane objects to the Panes collection, for a total of 4 panes:

```
SSSplitter1.Panes.Add "Pane A", 2    'split vertically  
SSSplitter1.Panes.Add "Pane B", 1    'split horizontally  
SSSplitter1.Panes.Add "Pane C", 1    'split horizontally  
  
MsgBox SSSplitter1.Panes.Count'display number of panes
```

PictureAnimationDelay Property Example

The following code increases the animation speed of an SSCommand button as the mouse is moved over it:

Place an SSCommand button on a form. Then place the following code into the **Load** event of the form:

```
Private Sub Form_Load()  
    SSCommand1.Width = 2000  
    SSCommand1.Height = 1500  
    SSCommand1.Caption = "Exit"  
    SSCommand1.Alignment = ssCenterBottom  
    SSCommand1.Picture = LoadPicture("C:\Sheridan\ActiveThreed\Graphics\  
Animations\Segmented Bitmaps\Red light(16).bmp")  
    SSCommand1.PictureMaskColor = vbBlue  
    SSCommand1.PictureAnimationDelay = 100  
    SSCommand1.PictureFrames = 16  
End Sub
```

Place the following code into the **MouseEnter** event of the SSCommand button:

```
Private Sub SSCommand1_MouseEnter(ByVal Button As Integer, ByVal Shift  
As Integer, ByVal X As Single, ByVal Y As Single)  
    SSCommand1.PictureAnimationDelay = 10  
End Sub
```

Place the following code into the **MouseExit** event of the SSCommand button:

```
Private Sub SSCommand1_MouseExit(ByVal Button As Integer, ByVal Shift As  
Integer, ByVal X As Single, ByVal Y As Single)  
    SSCommand1.PictureAnimationDelay = 100  
End Sub
```

Now run the program and move the mousepointer over the command button. You will notice that the animation speeds up as you pass the mouse over the control.

PictureAnimationEnabled Property Example

The following code animates an SSCommand button as the mouse is moved over it:

Place an SSCommand button on a form. Then place the following code into the **Load** event of the form:

```
Private Sub Form_Load()  
    SSCommand1.Width = 2000  
    SSCommand1.Height = 1500  
    SSCommand1.Caption = "Exit"  
    SSCommand1.Alignment = ssCenterBottom  
    SSCommand1.Picture = LoadPicture("C:\Sheridan\ActiveThreed\Graphics\  
Animations\Segmented Bitmaps\Red light(16).bmp")  
    SSCommand1.PictureMaskColor = vbBlue  
    SSCommand1.PictureAnimationDelay = 30  
    SSCommand1.PictureFrames = 16  
    SSCommand1.PictureAnimationEnabled = False  
End Sub
```

Place the following code into the **MouseEnter** event of the SSCommand button:

```
Private Sub SSCommand1_MouseEnter(ByVal Button As Integer, ByVal Shift  
As Integer, ByVal X As Single, ByVal Y As Single)  
    SSCommand1.PictureAnimationEnabled = True  
End Sub
```

Place the following code into the **MouseExit** event of the SSCommand button:

```
Private Sub SSCommand1_MouseExit(ByVal Button As Integer, ByVal Shift As  
Integer, ByVal X As Single, ByVal Y As Single)  
    SSCommand1.PictureAnimationEnabled = False  
End Sub
```

Now run the program and move the mouse pointer over the command button

Resize Event Example

This example creates a 2-way split window using the SSSplitter control. Each pane contains a label control displaying the pane's width.

First, Place one SSSplitter control and two label controls on a form. Then place the following code into the **Load** event of the form:

```
Private Sub Form_Load()  
    SSSplitter1.Panes.Add "Pane A", ssRightOfSplit  
    SSSplitter1.Panes(0).Control = Label1  
    SSSplitter1.Panes(1).Control = Label2  
  
    Label1.Caption = Int(SSSplitter1.Panes(0).Width /  
Screen.TwipsPerPixelX)  
    Label2.Caption = Int(SSSplitter1.Panes(1).Width /  
Screen.TwipsPerPixelX)  
End Sub
```

Place the following code into the **Resize** event of the SSSplitter control:

```
Private Sub SSSplitter1_Resize(ByVal BorderPanes As SSSplitter.Panes)  
    Label1.Caption = Int(SSSplitter1.Panes(0).Width /  
Screen.TwipsPerPixelX)  
    Label2.Caption = Int(SSSplitter1.Panes(1).Width /  
Screen.TwipsPerPixelX)  
End Sub
```

Now run the program and move the splitter bar between the two panes left and right. When you release the mouse button the labels are updated to display the width of both panes.

SplitterEndDrag Event Example

This example creates a 4-way split window using the SSSplitter control. Each pane contains a label control displaying the pane's name. Moving the splitter bar between panes A and B upward and downward will move panes B and C. Moving the splitter bar between panes B and D will not resize any panes.

First, place an SSSplitter control and 4 label controls on a form. Then place the following code into the **Load** event of the form:

```
Private Sub Form_Load()  
    Label1.Caption = "Pane A"  
    Label2.Caption = "Pane B"  
    Label3.Caption = "Pane C"  
    Label4.Caption = "Pane D"  
  
    SSSplitter1.Panes.Add "Pane A", 1  
    SSSplitter1.Panes.Add "Pane B", 1  
    SSSplitter1.Panes.Add "Pane B", 2  
  
    SSSplitter1.Panes(0).Control = Label1  
    SSSplitter1.Panes(1).Control = Label2  
    SSSplitter1.Panes(2).Control = Label3  
    SSSplitter1.Panes(3).Control = Label4  
  
    SSSplitter1.Panes(1).Height = 1000  
  
    SSSplitter1.AutoSize = ssAutoSizeFillContainer  
End Sub
```

Place the following code into the **SplitterEndDrag** event of the SSSplitter control:

```
Private Sub SSSplitter1_SplitterEndDrag()  
    SSSplitter1.Panes(1).Height = 1000 'resize Pane B  
End Sub
```

Place the following code into the **SplitterStartDrag** event of the SSSplitter control:

```
Private Sub SSSplitter1_SplitterStartDrag(ByVal SplitterBarType As Long,  
ByVal BorderPanels As Panels, Cancel As Boolean)  
    If ((BorderPanels(0).Control = Label3) And (BorderPanels(1).Control =  
Label4)) Then Cancel = True  
End Sub
```

Now run the program and move the splitter bar between panes A and B up and down. Without the code in the SplitterEndDrag event you would have moved the splitter bar, and as a result resized both panes B and C. By setting the height property of Pane B ("SSSplitter1.Pane(1)"), both panes B and C appear to move rather than resize. In addition, by placing the code in the SplitterStartDrag you can prevent the splitter bar between panes B and D from being dragged by the user. Although this could have been accomplished by setting Pane D's LockHeight property to True, it would have prevented the splitter from being moved as a result of the splitter between panes A and B being moved.

SplitterStartDrag Event Example

This example creates a 4-way split window using the SSSplitter control. Each pane contains a label control displaying the pane's name. Moving the splitter bar between panes A and B upward and downward will move panes B and C. Moving the splitter bar between panes B and D will not resize any panes.

First, place one SSSplitter control and four label controls on a form. Then place the following code into the **Load** event of the form:

```
Private Sub Form_Load()  
    Label1.Caption = "Pane A"  
    Label2.Caption = "Pane B"  
    Label3.Caption = "Pane C"  
    Label4.Caption = "Pane D"  
  
    SSSplitter1.Panes.Add "Pane A", 1  
    SSSplitter1.Panes.Add "Pane B", 1  
    SSSplitter1.Panes.Add "Pane B", 2  
  
    SSSplitter1.Panes(0).Control = Label1  
    SSSplitter1.Panes(1).Control = Label2  
    SSSplitter1.Panes(2).Control = Label3  
    SSSplitter1.Panes(3).Control = Label4  
  
    SSSplitter1.Panes(1).Height = 1000  
  
    SSSplitter1.AutoSize = ssAutoSizeFillContainer  
End Sub
```

Place the following code into the **SplitterEndDrag** event of the SSSplitter control:

```
Private Sub SSSplitter1_SplitterEndDrag()  
    SSSplitter1.Panes(1).Height = 1000 'resize Pane B  
End Sub
```

Place the following code into the **SplitterStartDrag** event of the SSSplitter control:

```
Private Sub SSSplitter1_SplitterStartDrag(ByVal SplitterBarType As Long,  
ByVal BorderPanes As Panes, Cancel As Boolean)  
    If ((BorderPanes(0).Control = Label3) And (BorderPanes(1).Control =  
Label4)) Then Cancel = True  
End Sub
```

Now run the program and move the splitter bar between panes A and B up and down. Without the code in the SplitterEndDrag event you would have moved the splitter bar, and as a result resized both panes B and C. By setting the height property of Pane B ("SSSplitter1.Pane(1)"), both panes B and C appear to move rather than resize. In addition, by placing the code in the SplitterStartDrag you can prevent the splitter bar between panes B and D from being dragged by the user. Although this could have been accomplished by setting Pane D's LockHeight property to True, it would have prevented the splitter from being moved as a result of the splitter between panes A and B being moved.

ssDataObjectFiles Collection Example

This example shows the use of the **Files** property to view and manipulate data contained in the **ssDataObjectFiles** collection (where "ssData" represents an object of type **ssDataObject**):

```
Debug.Print ssData.Files(index)
For Each v in ssData.Files
    Debug.Print v
Next v
ssData.Files.Add "autoexec.bat"
ssData.Files.Remove index
ssData.Files.Clear
For i = 0 to ssData.Files.Count - 1
    Debug.print ssData.Files(i)
Next i
```

Note This collection is used by the **Files** property only when the data in the **ssDataObject** object is in the ssCFFiles format (see [Constants](#) for more information.)

An **extender** property or method is a control property or method that is supplied and implemented by the host environment. For example, the **Left** and **Top** properties are not implemented within the control, so there may be host environments (for example, Visual C++) that don't have these properties available for the controls. Another example would be the **DataSource** and **DataField** properties which are implemented in Visual Basic but are not available in Delphi.

Entries marked with an asterisk are extender properties or methods and may not be available in all environments.

A property, such as **Picture** or **PictureDn** that specifies a picture to be displayed on the control, or a property, such as **PictureFrames** or **PictureAlignment** that affects how the specified picture(s) will be displayed.

Standard Event. This is a standard event found in Visual Basic controls. Consult your Visual Basic documentation for settings and other details on the use of this event.

Standard Method. This is a standard method found in Visual Basic controls. Consult your Visual Basic documentation for settings and other details on the use of this method. You may also use the Visual Basic Object Browser to view a brief description of this method.

Standard Property. This is a standard property found in Visual Basic controls. Consult your Visual Basic documentation for settings and other details on the use of this property. You may also use the Visual Basic Object Browser to view a brief description of this property.

See Also

[**LockHeight** property](#)

[**LockWidth** property](#)

[**Pane** object](#)

[**Panes** Collection](#)

See Also

[Height](#) property

[Pane](#) object

[Panes](#) collection

[Remove](#) method

[Width](#) property

See Also

[Alignment](#) property

[AutoSize](#) property (**SSplitter**)

See Also

[Align](#) property

[PictureAlignment](#) property

See Also

[**AutoSize** property \(SSPanel\)](#)

[**ButtonStyle** property](#)

[**Picture** property](#)

[**PictureDn** property](#)

See Also

AutoSize property (**SSCommand & SSRibbon**)

See Also

[Align](#) property

[SplitterResizeStyle](#) property

See Also

BackStyle property

ForeColor property

See Also

[**BackColor** property](#)

[**PictureBackground** property](#)

[**PictureBackgroundStyle** property](#)

See Also

BevelOuter property

BevelWidth property

BorderWidth property

See Also

BevelInner [property](#)

BevelWidth [property](#)

BorderWidth [property](#)

See Also

BevelInner [property](#)

BevelOuter [property](#)

BorderWidth [property](#)

See Also

Font property

Font object

See Also

AutoSize property

See Also

BevelInner property

BevelOuter property

See Also

BorderStyle property

See Also

Default [property](#)

See Also

[**Alignment** property](#)

[**CaptionStyle** property](#)

[**Font** property](#)

[**MarqueeStyle** property](#)

[**Picture** property](#)

See Also

[**MarqueeDirection** property](#)

[**MarqueeDelay** property](#)

[**MarqueeScrollAmount** property](#)

[**MarqueeStyle** property](#)

See Also

[**CheckBoxMaskColor** property](#)

[**CheckBoxUseMask** property](#)

[**OptionBtnGraphics** property](#)

[**Picture** property](#)

See Also

[**CheckBoxGraphics** property](#)

[**CheckBoxMaskColor** property](#)

[**CheckBoxUseMask** property](#)

See Also

CheckBoxGraphics property

CheckBoxMaskColor property

See Also

DbClick event

MouseDown event

MouseUp event

Value property (**SSCheck**)

Value Property (**SSCommand, SSOption, SSRibbon**)

See Also

ControlName property

Pane object

See Also

Control property

See Also

Panes collection

See Also

DataSource property

See Also

DataField [property](#)

See Also

[Click event](#)

[**MouseDown** event](#)

[**MouseUp** event](#)

See Also

[Cancel property](#)

See Also

[Click event](#)

See Also

OLEDragMode property

See Also

ssDataObject [object](#)

ssDataObjectFiles [collection](#)

See Also

[**BackColor** property](#)

[**FloodFillStyle** property](#)

[**FloodPercent** property](#)

[**FloodShowPct** property](#)

[**FloodType** property](#)

See Also

[**FloodColor** property](#)

[**FloodPercent** property](#)

[**FloodShowPct** property](#)

[**FloodType** property](#)

See Also

[**FloodColor** property](#)

[**FloodFillStyle** property](#)

[**FloodShowPct** property](#)

[**FloodType** property](#)

See Also

FloodColor [property](#)

FloodPercent [property](#)

FloodType [property](#)

See Also

[**FloodColor** property](#)

[**FloodFillStyle** property](#)

[**FloodPercent** property](#)

[**FloodShowPct** property](#)

See Also

[Font property](#)

See Also

Font object

Compatibility Issues

See Also

[Caption property](#)

See Also

BackColor property

See Also

[**GetFormat** method](#)

[**OLESetData** event](#)

[**SetData** method](#)

[**ssDataObject** object](#)

See Also

[**GetData** method](#)

[**OLESetData** event](#)

[**SetData** method](#)

[**ssDataObject** object](#)

See Also

GroupNumber property

See Also

GroupAllowAllUp property

See Also

[Left property](#)

[Top property](#)

[Width property](#)

See Also

[Font property](#)

[Font object](#)

See Also

KeyPress event

KeyUp event

See Also

KeyDown event

KeyUp event

See Also

KeyUp event

KeyDown event

See Also

Height property

Top property

Width property

See Also

Locked property

LockWidth property

See Also

Locked property

LockWidth property

See Also

LockHeight [property](#)

LockWidth [property](#)

See Also

[**MarqueeCycleEnd** event](#)

[**MarqueeDelay** property](#)

[**MarqueeDirection** property](#)

[**MarqueeStyle** property](#)

See Also

[**MarqueeCycleBegin** event](#)

[**MarqueeDelay** property](#)

[**MarqueeDirection** property](#)

[**MarqueeStyle** property](#)

See Also

[**CaptionStyle** property](#)

[**MarqueeDirection** property](#)

[**MarqueeScrollAmount** property](#)

See Also

CaptionStyle [property](#)

MarqueeCycleBegin [event](#)

See Also

[**CaptionStyle** property](#)

[**MarqueeDelay** property](#)

[**MarqueeDirection** property](#)

See Also

[**CaptionStyle** property](#)

[**MarqueeCycleBegin** event](#)

[**MarqueeDirection** property](#)

[**MarqueeDelay** property](#)

[**MarqueeScrollAmount** property](#)

See Also

LockHeight property

MinWidth property

See Also

LockWidth [property](#)

MinHeight [property](#)

See Also

[Click event](#)

[DblClick event](#)

[MouseUp event](#)

See Also

MouseExit event

MouseMove event

See Also

[MouseEnter](#) event

[MouseMove](#) event

See Also

MousePointer property

See Also

MouseEnter event

MouseExit event

See Also

MouseIcon property

See Also

[Click event](#)

[DblClick event](#)

[MouseDown event](#)

See Also

[Font property](#)

See Also

[**OLEDrag** method](#)

[**OLEDragDrop** event](#)

[**OLEDragOver** event](#)

[**OLEDropMode** property](#)

[**OLEGiveFeedback** event](#)

[**OLESetData** event](#)

[**OLEStartDrag** event](#)

See Also

[**OLECompleteDrag** event](#)

[**OLEDragDrop** event](#)

[**OLEDragOver** event](#)

[**OLEDropMode** property](#)

[**OLEGiveFeedback** event](#)

[**OLESetData** event](#)

[**OLEStartDrag** event](#)

See Also

[**OLECompleteDrag** event](#)

[**OLEDrag** method](#)

[**OLEDragOver** event](#)

[**OLEDropMode** property](#)

[**OLEGiveFeedback** event](#)

[**OLESetData** event](#)

[**OLEStartDrag** event](#)

See Also

[**OLECompleteDrag** event](#)

[**OLEDrag** method](#)

[**OLEDragDrop** event](#)

[**OLEDropMode** property](#)

[**OLEGiveFeedback** event](#)

[**OLESetData** event](#)

[**OLEStartDrag** event](#)

See Also

[**OLECompleteDrag** event](#)

[**OLEDrag** method](#)

[**OLEDragDrop** event](#)

[**OLEDragOver** event](#)

[**OLEGiveFeedback** event](#)

[**OLESetData** event](#)

[**OLEStartDrag** event](#)

See Also

[**OLECompleteDrag** event](#)

[**OLEDrag** method](#)

[**OLEDragDrop** event](#)

[**OLEDragOver** event](#)

[**OLEDropMode** property](#)

[**OLESetData** event](#)

[**OLEStartDrag** event](#)

See Also

[**OLECompleteDrag** event](#)

[**OLEDrag** method](#)

[**OLEDragDrop** event](#)

[**OLEDragOver** event](#)

[**OLEDropMode** property](#)

[**OLEGiveFeedback** event](#)

[**OLEStartDrag** event](#)

[**SetData** method](#)

[**ssDataObject** object](#)

See Also

[**OLECompleteDrag** event](#)

[**OLEDrag** method](#)

[**OLEDragDrop** event](#)

[**OLEDragOver** event](#)

[**OLEDropMode** property](#)

[**OLEGiveFeedback** event](#)

[**OLESetData** event](#)

See Also

[**CheckBoxGraphics** property](#)

[**OptionBtnMaskColor** property](#)

[**OptionBtnUseMask** property](#)

[**Picture** property](#)

See Also

[**OptionBtnGraphics** property](#)

[**OptionBtnUseMask** property](#)

See Also

OptionBtnGraphics [property](#)

OptionBtnMaskColor [property](#)

Properties

Events

Methods

Objects

Collections

See Also

[Font object](#)

[Panels collection](#)

See Also

PaneFromPosition method

See Also

PaneFromControl method

PaneFromPositionEx method

See Also

PaneFromControl method

PaneFromPosition method

See Also

[Pane object](#)

See Also

[**PictureAlignment** property](#)

[**PictureBackground** property](#)

[**PictureFrames** property](#)

See Also

Alignment [property](#)

Picture [property](#)

See Also

[PictureFrames](#) property

See Also

PictureAnimationDelay property

See Also

BackStyle property

Picture property

See Also

[**BackColor** property](#)

[**BackStyle** property](#)

[**PictureBackground** property](#)

See Also

[Picture](#) property

[PictureDisabledFrames](#) property

[PictureDn](#) property

See Also

[Picture](#) property

[PictureDisabled](#) property

[PictureFrames](#) property

See Also

[Picture](#) property

[PictureDisabled](#) property

[PictureDnFrames](#) property

See Also

[Picture](#) property

[PictureDn](#) property

See Also

[Picture](#) property

[PictureDisabled](#) property

[PictureDn](#) property

See Also

[Picture](#) property

[PictureDnDisabled](#) property

[PictureFrames](#) property

See Also

[Picture](#) property

[PictureDn](#) property

[PictureFrames](#) property

See Also

Picture property

See Also

PictureUseMask property

See Also

[Picture](#) property

[PictureMaskColor](#) property

See Also

Locked property

See Also

ButtonStyle property

See Also

[**OLESetData** event](#)

[**GetData** method](#)

[**ssDataObject** object](#)

See Also

BorderStyle property

See Also

[Font object](#)

[Font property](#)

See Also

SplitterBarWidth [property](#)

SplitterBarJoinStyle [property](#)

See Also

[SplitterBarAppearance](#) property

See Also

[SplitterBarAppearance](#) property

See Also

SplitterStartDrag event

See Also

[AutoSize](#) property

[SplitterBarAppearance](#) property

See Also

SplitterEndDrag event

See Also

[Font object](#)

[Font property](#)

See Also

[Height property](#)

[Left property](#)

[Width property](#)

See Also

Value property (SSCheck)

Upgrade Notes

See Also

[Font object](#)

[Font property](#)

See Also

[TripleState](#) property

[Value](#) property (**SSCommand, SSOption, SSRibbon**)

See Also

[DoClick](#) method

[Value](#) property (**SSCheck**)

See Also

[Height property](#)

[Left property](#)

[Top property](#)

See Also

[Windowed vs. Windowless Operation](#)

See Also

[**ssDataObjectFiles** collection](#)

[**OLECompleteDrag** event](#)

[**OLEDrag** method](#)

[**OLEDragDrop** event](#)

[**OLEDragOver** event](#)

[**OLEDropMode** property](#)

[**OLEGiveFeedback** event](#)

[**OLESetData** event](#)

[**OLEStartDrag** event](#)

See Also

[**ssDataObject** object](#)

[**OLECompleteDrag** event](#)

[**OLEDrag** method](#)

[**OLEDragDrop** event](#)

[**OLEDragOver** event](#)

[**OLEDropMode** property](#)

[**OLEGiveFeedback** event](#)

[**OLESetData** event](#)

[**OLEStartDrag** event](#)

ssDataObject Object

[See Also](#)

[Applies To](#)

Description

An **ssDataObject** object is a container for data being transferred from an component source to an component target. The data is stored in the format defined by the method using the **ssDataObject** object.

Properties

[Files](#)

Methods

[Clear](#)

[GetData](#)

[GetFormat](#)

[SetData](#)

Collections

[ssDataObjectFiles](#)

Remarks

The **ssDataObject**, which mirrors the Visual Basic DataObject object and the IDataObject interface, allows OLE drag and drop and clipboard operations to be implemented.

ssDataObjectFiles Collection

[See Also](#)

[Example](#)

[Applies To](#)

Description

A collection of strings which is the type of the Files property on the **ssDataObject** object.

Properties

[Count](#)

Methods

[Clear](#)

[Item](#)

[Remove](#)

Remarks

The **ssDataObjectFiles** collection is a collection of strings which represent a set of files which have been selected either through the **GetData** method, or through selection in an application such as the Windows Explorer.

Although the **ssDataObjectFiles** collection has methods and properties of its own, you should use the Files property of the ssDataObject object to view and manipulate the contents of the **ssDataObjectFiles** collection.

